VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Optimalizace uživatelského rozhraní mobilního informačního systému myAVIS

The Mobile Information System myAVIS user Interface Optimalization

2016 Martin Kovář

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání diplomové práce

Student: **Bc. Martin Kovář**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2612T025 Informatika a výpočetní technika

Téma: Optimalizace uživatelského rozhraní mobilního informačního systému myAVIS
The Mobile Information System myAVIS user Interface Optimalization

Jazyk vypracování: čeština

Zásady pro vypracování:

Cílem diplomové práce je optimalizace a implementace uživatelského rozhraní aplikace pro mobilní zařízení.

1. Analýza současného stavu uživatelského rozhraní mobilního informačního systému myAVIS - definice požadavků z hlediska ergonomie a komfortu ovládání uživatele.
2. Rešerše možností implementace využití komerčních komponent - Sencha Touch, Telerik Mobile UI, Infragistics Android/iOS/MobilePhone controls.
3. Možnosti implementace standardu HTML5 v rámci vývoje mobilních aplikací.
4. Implementace dynamických controls při implementaci uživatelských rozhraní mobilních aplikací - našeptávače, autocomplete dropdown menu, dynamické filtry apod.
5. Návrh cross-platform UI.

Seznam doporučené odborné literatury:

Podle pokynů vedoucího diplomové práce.

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Ing. Radek Garzina, Ph.D.**

Konzultant diplomové práce: Mgr. Ing. Michal Krumnikl, Ph.D.

Datum zadání: 01.09.2014

Datum odevzdání: 29.04.2016

doc. Dr. Ing. Eduard Sojka	prof. RNDr. Václav Snášel, CSc.
vedoucí katedry	*děkan fakulty*

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 1.dubna 2016

.....

Souhlasím se zveřejněním této diplomové práce dle požadavků čl. 26, odst. 9 Studijního a zkušebního řádu pro studium v magisterských programech VŠB-TU Ostrava.

V Ostravě 1.dubna 2016

.....

Rád bych poděkoval vedoucímu mé diplomové práce, Ing. Radku Garzinovi, Ph.D. za jeho ochotu, trpělivost při konzultacích a perfektní vedení. Dále bych rád poděkoval mým rodičům a mé přítelkyni, kteří mě během studia plně podporují.

Abstrakt

Diplomová práce se zabývá analýzou současného stavu uživatelského rozhraní produktu myAvis NG, který vyvinula společnost Kvados. Tato diplomová práce se věnuje optimalizaci uživatelského rozhraní mobilního informačního systému myAvis NG. Práce shrnuje a popisuje možnosti implementace mobilních aplikací a předvádí způsob implementace klientských aplikací, který se používá ve společnosti Kvados. V rámci diplomové práce je vytvořena řada komponent uživatelského rozhraní, které jsou určeny pro mobilní informační systém myAvis NG.

Klíčová slova: Kvados, myAvis, Android, Java, optimalizace, uživatelské rozhraní, mobilní informační systém, komponenta, aplikace, LaTeX

Abstract

Thesis deals with analysis the current state of the user interface of the product myAvis NG which is developed by the company Kvados. This thesis focuses on optimization of the user interface on the mobile information system myAvis NG. The thesis summarizes possibilities of mobile applications development and demonstrates how to develop client applications in the way, which is used in the company Kvados. We developed several user interface components that are designed for mobile information system myAvis NG.

Key Words: Kvados, myAvis, Android, Java, optimization, user interface, mobile information system, component, application, LaTeX

Obsah

Seznam použitých zkratek a symbolů

UI	–	Uživatelské rozhraní (User Interface)
CRM	–	Řízení vztahů se zákazníky (Customer Relationship Management)
SFA	–	Automatizace pracovních úkolů (Sales Force Automation)
ERP	–	Plánování podnikových zdrojů (Enterprise Resource Planning)
MIS	–	Manažerský informační systém (Management Information System)
XML	–	Extensible Markup Language
HDML	–	Handheld Device Markup Language
HTML	–	Hypertext Markup Language
CSS	–	Cascading Style Sheets
dpi	–	Počet pixelů na palec (Dots Per Inch)
SDK	–	Sada vývojových nástrojů (Software Development Kit)
IDE	–	Vývojové prostředí (Integrated Development Environment)
TFS	–	Sada nástrojů pro týmovou spolupráci (Team Foundation Server)
SCSF	–	Smart Client Software Factory
API	–	Application Programming Interface
iOS	–	Mobilní operační systém vytvořený společností Apple

Seznam obrázků

Seznam výpisů zdrojového kódu

1 Úvod

Tématem diplomové práce je optimalizace uživatelského rozhraní mobilního informačního systému myAvis NG. Diplomová práce se zabývá analýzou současného stavu uživatelského rozhraní produktu, který je určený pro operační systém Android. V této kapitole představím společnost Kvados, produkt myAvis NG, architekturu řešení myAvis a operační systém Android. Následně v teoretické části shrnu možnosti implementace mobilních aplikací, zatímco v implementační části se budu převážně věnovat tvorbě komponent uživatelského rozhraní a ovládacích prvků.

Optimalizace uživatelského rozhraní je nezbytnou součástí každé moderní aplikace. Komfort při ovládání produktu myAvis NG je velice důležitý. Uživatelské rozhraní je navrženo tak, aby uživatel rychle dosáhl požadovaného výsledku. Při návrhu uživatelského rozhraní je nutné zvážit použití a umístění ovládacích prvků tak, aby uživatel strávil zadáváním a prohlížením dat minimální množství času. Optimalizované uživatelské rozhraní šetří každé zbytečné kliknutí, vede uživatele při zadávání dat a zobrazuje veškerá potřebná data.

V rámci této diplomové práce jsem vytvořil komponenty *RecyclerViewControl* a *DataQueryAutoCompleteTextView*, které se nyní používají v řadě zákaznických projektů. Během mého působení ve společnosti Kvados jsem se podílel na spoustě dalších komponent uživatelského rozhraní, jako jsou například *ListViewControl*, *VerticalTextView*, *MultiSpinner* apod. Vzhledem k tomu, že se jednalo pouze částečně o moji práci, tak ji zde nebudu více popisovat.

Dále jsem se podílel na návrzích uživatelského rozhraní pro různé agendy aplikace myAvis NG. Působil jsem jako konzultant a software designer při návrhu uživatelského rozhraní a následně jako její tvůrce.

Navrhnul a vytvořil jsem zcela nové ovládací prvky umožňující práci s kalendářem v denním, týdenním a měsíčním režimu.

V rámci této diplomové práce jsem se také zaměřil na podporu více typů rozlišení a velikostí obrazovky. Podpora různých obrazovek se ukázala jako nezbytná, protože zákazníci společnosti Kvados, kteří si koupili aplikaci myAvis NG, aktuálně používají různá zařízení, jako jsou například Lenovo Yoga, Samsung Galaxy S6 apod.

V celém textu budeme používat výraz „rozvržení“ místo anglického výrazu „layout“, výraz „zdroj“ místo anglického výrazu „resource“ a výraz „spoušť“ místo anglického výrazu „trigger“. Výrazy *RootWorkItem*, *WorkItem*, *Panel*, *ViewControl*, *ControlItem*, *Command*, *ControlData* a *ToolItem* reprezentují třídy a v celém textu je budu skloňovat podle vzoru hrad.

Diplomová práce je rozdělena na veřejnou a neveřejnou část. V neveřejné části diplomové práce se vyskytují pouze snímky obrazovky pořízené v aplikaci myAvis NG. Snímky jsou umístěny v neveřejné části z důvodu ochrany firemního know-how. Obrázky jsou popsány a vysvětleny v této části práce a je vždy jasně specifikováno, že se jedná o obrázek umístěný v neveřejné části diplomové práce.

1.1 Kvados

Kvados je významným producentem a dodavatelem vlastních softwarových řešení, která jsou známá pod značkami Ventus, myAvis, myCash, myWork, myFaber, myTeam, myStock a myMachine.

Na trhu působí od roku 1992 a zaměřuje se především na klienty ze segmentu obchodu a služeb. Společnost si zakládá na vysoké kvalitě vnitřních procesů, které má certifikovány dle všech 6 norem využitelných v oblasti informačních a komunikačních technologií.

Kvados dlouhodobě vyhledává příležitosti na nových trzích, což se podařilo například u mobilních informačních systémů. Softwarová řešení společnosti přispěly k lepším hospodářským výsledkům a k optimalizaci procesů zákazníků.

Kvados je českou nezávislou akciovou společností s centrálou v Ostravě. Společnost se zapojuje do lokálního dění a podporuje kvalitní projekty. Jedná se o zakládajícího člena IT Clusteru, což je sdružení, které je vůdčí silou výzkumu a vývoje nových technologií v Moravskoslezském kraji. Kvados investuje do vlastního výzkumu až 15% obratu ročně a spolupracuje s dalšími předními mezinárodními společnostmi. Nejen vývoj, ale také provoz informačních systémů a služeb odpovídá nejnáročnějším požadavkům dnešní doby. Dokládají to certifikáty mezinárodních standardů ISO:

1. ISO 9001:2009 (management kvality),
2. ISO 10006:2004 (řízení projektů),
3. ISO 14001:2005 (environmentální management),
4. ISO 18001:2008 (systém řízení bezpečnosti a ochrany zdraví),
5. ISO 20000-1:2006 (poskytování služeb IT),
6. ISO 27001:2006 (řízení informační bezpečnosti).

1.2 myAVIS NG

Řešení myAvis NG navazuje na mnohaleté zkušenosti společnosti Kvados s vývojem a implementací mobilních informačních systémů. Přináší nový rozměr do práce uživatelů v terénu. Řešení rozšiřuje nástroj myAvis určený pro mobilní platformu Microsoft Windows a přebírá z něj rozsáhlou funkčnost.

Jedná se o unikátní CRM nástroj specificky navržený a implementovaný pro využití na tabletech a mobilních telefonech s operačním systémem Android. Více informací o operačním systému Android se nachází v kapitole 1.3.

Procesy a agendy myAvis NG vycházejí z logiky obchodníků. Aplikace přináší nejen rozsáhlé možnosti ke komplexní prezentaci zboží a služeb, ale také k pohodlnému a snadnému pořizování a zpracování dat. Do vztahů se zákazníky dává nový rozměr, aktuálnost a přehlednost pro všechny

uživatele. Umožňuje evidovat všechny aktivity a komunikaci se zákazníkem. Dokáže sdružit a integrovat do sebe data i z ostatních systémů.

Řešení myAvis NG mohou využívat nejen obchodní zástupci, ale také celý obchodní tým, včetně nejvyššího vedení. Díky rozsáhlejším možnostem vizualizace se obchodníkům zobrazují informace dříve dostupné pouze manažerům. Pomocí grafů a diagramů mohou sledovat, jak se jednotliví zákazníci podílejí na tržbách, zobrazit si graf neuhrazených faktur s intervalem po splatnosti nebo vývoj platební morálky. Další údaje mohou porovnávat v různých žebříčcích. Samozřejmá je podpora videa a fotografií, například pro ukázku reklamních materiálů během obchodní schůzky.

Přes širokou funkčnost je aplikace myAvis NG navržena pro jednoduché ovládání. Vše funguje tak, jak uživatel předpokládá. Přirozené a intuitivní ovládání šetří každé zbytečné kliknutí. Uživatel by měl strávit zadáváním a prohlížením dat minimální množství času. Toho je dosaženo pečlivým a promyšleným výběrem ovládacích prvků, jejich grafickým a gestickým zpracováním, volbou informací a zvýrazňováním pomocí barev. Zároveň tím myAvis NG definuje ve své třídě nové standardy tabletových aplikací.

1.2.1 Filozofie řešení

V minulých letech společnost Kvados úspěšně implementovala produkt myAvis desítkám spokojených zákazníků. Je využíván tisíci koncovými uživateli, kteří jej dennodenně využívají ke své práci. S jeho pomocí zákazníci dosahují pracovních výsledků v požadované kvalitě za minimální nezbytný čas.

Produkt myAvis NG přináší nové možnosti. Je dlouhodobě vyvíjen tak, aby se obsluhoval, pracoval a fungoval způsobem, jakým přemýšlejí koncoví uživatelé. Nová verze je od samotného začátku navržena tak, aby zohledňovala pohled obchodníků, nikoliv programátorů. Při terénní práci s myAvis NG jsou koncoví uživatelé rychlejší, informovanější a efektivnější. Přirozený a soustavný tlak zákazníků je motivací, jak udělat řešení myAvis NG lepší a více přizpůsobený pro práci v terénu.

Společnost Kvados věnovala značné úsilí grafickému designu. Bylo navrženo nové unikátní grafické rozhraní. Výsledkem je kompromis mezi krásou designu, účelností a ovladatelností.

Při vývoji produktu myAvis NG se vybralo to nejlepší ze zkušeností, znalostí a dovedností získaných za poslední desetiletí při vývoji produktu myAvis. K tomu se doplnil nový přístup a technologie. Změnil se způsob a logika licencování a zároveň se kriticky posoudily předešlé obchodní modely.

1.2.2 Popis řešení

Společnost Kvados dlouho hledala řešení, jak změnit aplikaci na tabletu, která se většinově hodí na veškerou práci, je založena na principu konzumace obsahu, a také je značně neefektivní pro práci při pořizování dat a informací. Výsledkem náročné práce je návrh nového rozhraní, ve

kterém je práce snadná, informace jsou přehledně a čistě zobrazeny a vkládání dat je pohodlné a efektivní. Aplikace poskytuje uživateli přímo do terénu informace, které dříve měli k dispozici pouze manažeři v kancelářích. Řešení je navrženo tak, aby práce každého uživatele byla vždy perfektní, cílevědomá a podpořena systematickou přípravou na jednání se zákazníkem. Důsledně vede uživatele všemi obchodními procesy a aktivitami. Dohlíží na dodržování nastavených pravidel, postupů a vynutí si jejich dodržování. Systém práce přitom vychází z uvažování a logiky dobrých a úspěšných obchodníků tak, jak je definovali zkušení vedoucí obchodních týmů a obchodní ředitelé.

1.2.3 Architektura řešení

Řešení myAvis se skládá z několika nedílných součástí, které dohromady zajišťují nejen chod celého systému, ale poskytují i navazující funkce, jako je napojení na externí systémy typu ERP, MIS apod.

1. **Mobilní klientská aplikace myAvis NG** je základním stavebním kamenem celého systému myAvis. Slouží pro okamžitý sběr a konzumaci dat pracovníky v terénu pomocí mobilních zařízení se systémem Android.

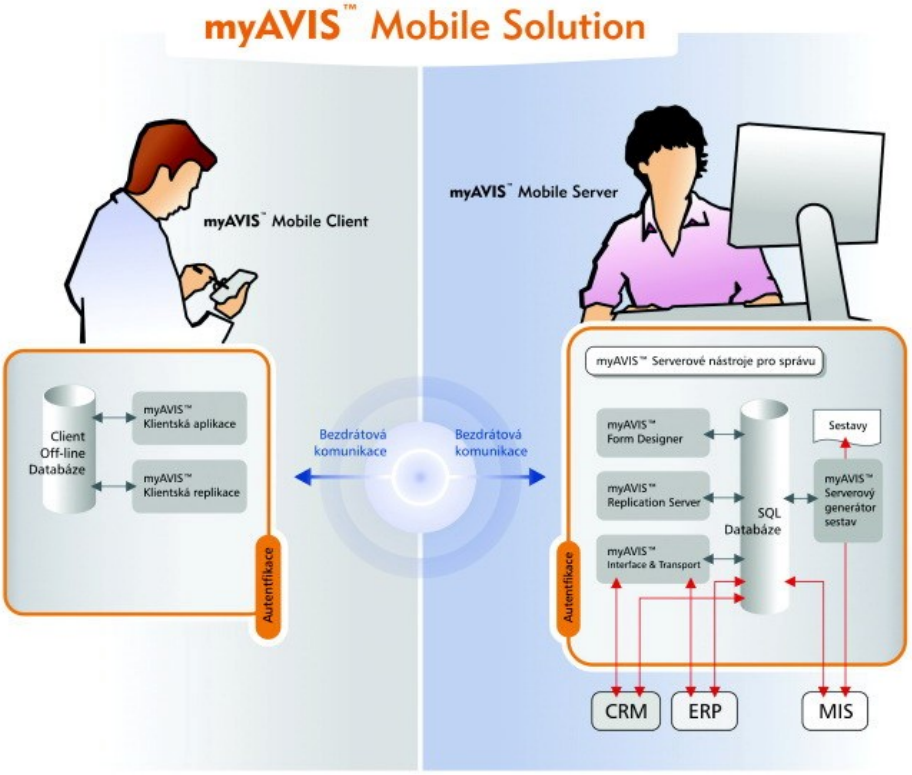
2. **Přenosová technologie myAvis NG** zajišťuje síťovou komunikaci mezi klientskou a serverovou částí systému. Hlavní funkcí je komunikace mezi klientskou aplikací myAvis NG a serverovou částí myAvis NG Mobile Server. Technologie je založená na využití IP protokolu (GSM, modem, LAN, WAN, GPRS, EDGE apod.).

3. **Serverová část myAvis NG Mobile Server**, kde jsou všechna pořízená data shromažďována, tříděna a distribuována do dalších systémů. Využívá technologii Microsoft SQL Server. Součástí je také robustní nástroj pro administraci celého řešení.

1.2.3.1 Mobilní klientská aplikace myAvis NG Hlavní rolí mobilní klientské aplikace myAvis NG je poskytovat oporu uživatelům při provádění obchodních procesů v terénu. Všechna data, která během procesů vznikají, jsou zaznamenávána do lokální databáze, která je pravidelně synchronizována s aktuálními daty na serveru. Aplikaci je možné používat i v offline režimu. S každou synchronizací aplikace odesílá pořízená data a aktualizuje data, která byla na serveru změněna jinými uživateli.

1.3 Android

Android je všude. Najdete jej v telefonech, tabletech, televizích a set-top boxech s technologií Google TV a brzy bude i v robotech, automobilech, dokonce i v zábavních systémech letadel. Hlavní oblastí uplatnění systému Android však budou i nadále zařízení s menšími obrazovkami a s hardwarovou klávesnicí či bez ní. Systém Android je spojován především s chytrými telefony a tablety.



Obrázek 1: Architektura řešení.

Android je flexibilní operační systém, který je využíván na více než miliardě zařízení po celém světě. V současné době je zastřešován společností Google a je založen na linuxovém jádře. Jedná se o otevřenou platformu, zdrojové kódy jsou k dispozici, a to od vysokoúrovňových aplikačních rámců, přes nativní knihovny až po nízkoúrovňové linuxové moduly. Toto umožňuje výrobcům zařízení si operační systém do jisté míry upravit, což se také často děje.

Systém Android je licencován bezplatně a nejsou stanoveny žádné speciální požadavky na hardware, který operační systém používá. Platforma se stala v posledních letech velice populární a hlavním důvodem je právě bezplatná licence.

Hlavní výhodou zařízení se systémem Android je jejich přitažlivost. Možnost používání internetových služeb v mobilních zařízeních je zde již od poloviny devadesátých let a vzniku jazyka HDML. Opravdový boom telefonů umožňujících přístup k Internetu však nastal teprve v posledních letech. Díky trendům, jako je zasílání online textových zpráv nebo zařízení iPhone společnosti Apple, popularita telefonů, které lze použít jako přístupový bod k Internetu, rychle vzrůstá. Systém Android se uplatňuje na rychle rostoucím segmentu trhu - telefony s připojením k Internetu.

Produkt myAvis NG je určen pro operační systém Android, protože jej používá většina mobilních zařízení. Další výhodou je lepší cenová dostupnost mobilních zařízení s operačním systémem Android.

1.3.1 Historie

Operační systém byl vyvinut společností Android, která byla založena v říjnu roku 2003 v Kalifornii. Tuto téměř neznámou společnost odkoupila společnost Google v srpnu roku 2005, která se o vývoj operačního systému stará dodnes. V říjnu roku 2008 byl uveden první telefon s operačním systémem Android a současně byla uvolněna první verze Android SDK.

1.4 Cíl práce

V této kapitole jsem představil společnost Kvados, produkt myAvis NG a architekturu řešení myAvis. Kapitolu jsem zakončil představením operačního systému Android.

V následující kapitole 2 si ukážeme současný stav uživatelského rozhraní produktu myAvis NG. Budeme se zabývat zadáváním dat, gesty a rozdělením obrazovky. Ukážeme si některá typická zobrazení, jako jsou například seznam zákazníků nebo editace ceníku.

Při vývoji mobilní aplikace se můžeme vydat různými směry, které určují vznik a možnosti aplikace. Kapitola 3 shrnuje a popisuje možnosti implementace mobilních aplikací. Shrneme výhody a nevýhody uvedených možností vývoje mobilních aplikací. Představíme si platformu PhoneGap založenou na standardu HTML5, která umožňuje vývoj aplikací na základě jednotné technologie a jejich nasazení na více různých platformách.

V kapitole 4 si ukážeme multiplatformní vývoj používaný ve společnosti Kvados, který vychází z principů a tříd používaných pro psaní WinForm SmartClient aplikací. Předvedeme si AndroidClient aplikaci a objekt zvaný *WorkItem*.

Tyto informace využijeme v kapitole 5, ve které si probereme implementaci komponent uživatelského rozhraní pro AndroidClient aplikaci. Ukážeme si implementaci *ViewControl* komponenty, *ControlItem* komponenty a vlastních ovládacích prvků.

2 Analýza současného stavu uživatelského rozhraní

Aplikace myAvis NG je určena pro zařízení s operačním systémem Android, který poskytuje obrovské možnosti pro zdokonalování na poli ergonomie a intuitivnosti. Právě tyto vlastnosti společně s jednoduchostí jsou hlavními znaky aplikace. Velké displeje tabletů umožňují aplikaci zobrazovat veškerá potřebná data komfortně a rychle, nejen v textové formě. V aplikaci jsou proto samozřejmostí grafy, videa, fotografie apod.

Veškeré ovládání aplikace je navrženo tak, aby uživatel rychle dosáhl požadovaného výsledku. Proto jsou jednotlivé ovládací prvky, jako například tlačítka, textová pole ve formulářích, položky seznamů apod., dostatečně velké. Všechny obrazovky aplikace jsou navíc koncipovány tak, že je možné je jednoduše přeskládat v případě, že uživatel je levák.

2.1 Zadávání dat

Velký důraz je v aplikaci myAvis NG kladen na zadávání dat. Rutinní činnosti, jako je zadávání textů nebo číselných hodnot, jsou optimalizovány tak, aby s nimi uživatel strávil minimum času. Snažíme se ovládací prvky umisťovat k pravé a levé části obrazovky tak, aby se uživatel nemusel trápit při zadávání dat. Pokud uživatel drží zařízení oběma rukama, může jednoduše palci dosáhnout na každý ovládací prvek.

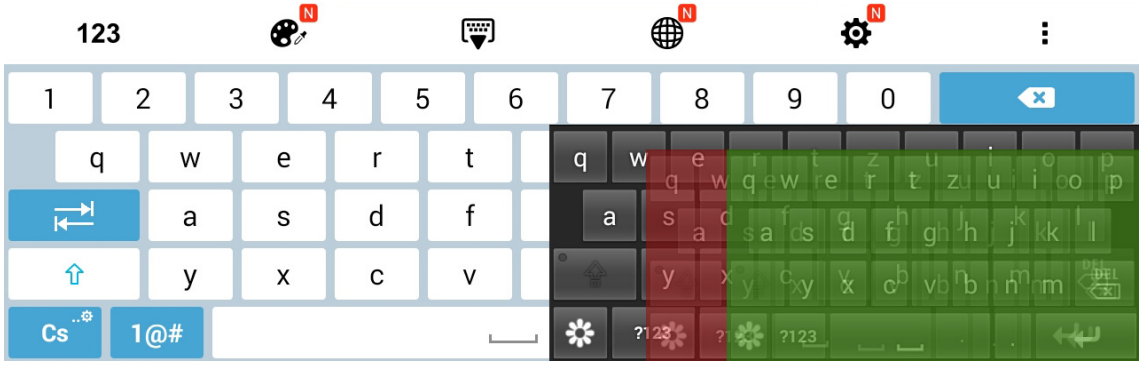
Systémová klávesnice obvykle zabírá až polovinu obrazovky, což způsobuje snížení přehlednosti při zadávání dat. Byla vytvořena nová klávesnice, která zabírá jen malou část obrazovky. Umístění a šířka klávesnice je navíc nastavitelná podle potřeb uživatele. Klávesnice je navržena tak, aby ji bylo možné používat jednou rukou, zatímco druhou rukou můžeme ovládat například seznam v pracovní ploše. Zadávání dat je tedy jednoduché a příjemné.

Porovnání velikostí QWERTZ a numerických klávesnic je na obrázku 2 a 3. Na obrázku 1 neveřejné části je klávesnice standardně umístěna v pravém dolním rohu, zatímco na obrázku 2 neveřejné části je klávesnice přesunuta do levého dolního rohu.

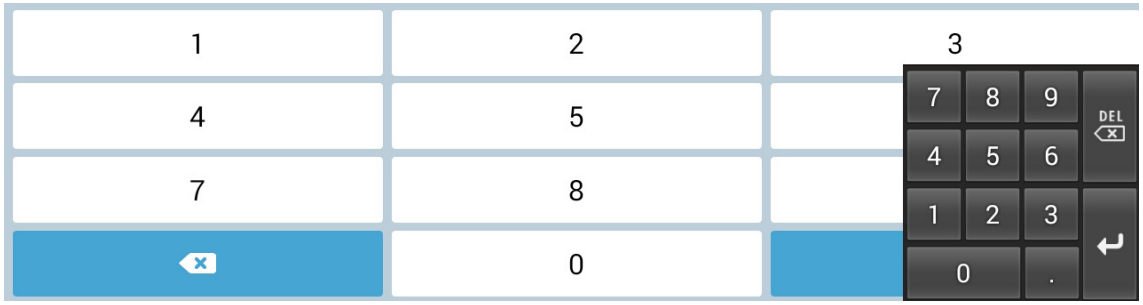
2.2 Gesta

Operační systém Android poskytuje velkou podporu takzvaným gestům. Gesta jsou specifické pohyby prsty na displeji tabletu intuitivně kopírující význam akcí, které jsou jimi spouštěny. Aplikace myAvis NG nepodporuje žádná speciální gesta. V celé aplikaci si uživatel vystačí s posouváním, jednoduchým a dlouhým kliknutím.

Zvláštní pozornost byla věnována otázce, zda uživateli umožnit zoomování pracovní plochy. Zoomování je možné pouze v mapě, kalendáři a galerii obrázků. Obrazovky jsou navrženy takovým způsobem, aby zoomování nebylo vůbec potřeba. Zoomování totiž snižuje přehlednost a komplikuje ovládání.



Obrázek 2: Porovnání velikostí QWERTZ klávesnic.



Obrázek 3: Porovnání velikostí numerických klávesnic.

2.3 Klíčové přístupy

Produkt myAvis NG především podporuje procesy zákazníka, a proto práce s aplikací podléhá těmto procesům. Aplikace vede uživatele při vyplňování dat, tudíž je ovládání přehledné, jednoduché a rychlé.

Rozhraní je propracované, ale čisté, jednoduché, srozumitelné, přehledné, spíše funkcionální. Aplikace neobsahuje žádná složitá tlačítka a další komplikované ovládací prvky. Barvy jsou velice střídmé, barevně je v aplikaci zvýrazněno pouze to, co je potřeba zdůraznit.

Komfort ovládání je důležitější než množství zobrazených informací. Z toho důvodu je na každé obrazovce přítomen stavový panel a hlavní panel akcí. Přítomnost stavového panelu a hlavního panelu akcí sice zmenšuje pracovní plochu, ale zvyšuje rychlost ovládání a přehlednost. Jedním dotykem lze například smazat smlouvu, stornovat objednávku nebo vygenerovat fakturu.

Produkt myAvis NG především podporuje horizontální pohled. Hlavní důvod pro toto rozhodnutí je návrh pracovní plochy aplikace, která většinou obsahuje seznam položek, detail vybrané položky a postranní panel, který obsahuje užitečné informace o dané obrazovce. Drtivá většina zákazníků volí pro své obchodní zástupce především tablety nebo větší chytré telefony.

Při návrhu aplikace se navíc předpokládalo, že displeje u mobilních zařízení se budou zvětšovat. Tento předpoklad se ukázal jako pravdivý. Stále se zvyšuje počet mobilních zařízení, na

kterých je možné aplikaci efektivně používat. Trend zvětšování displejů mobilních zařízení je předveden na obrázku 4.

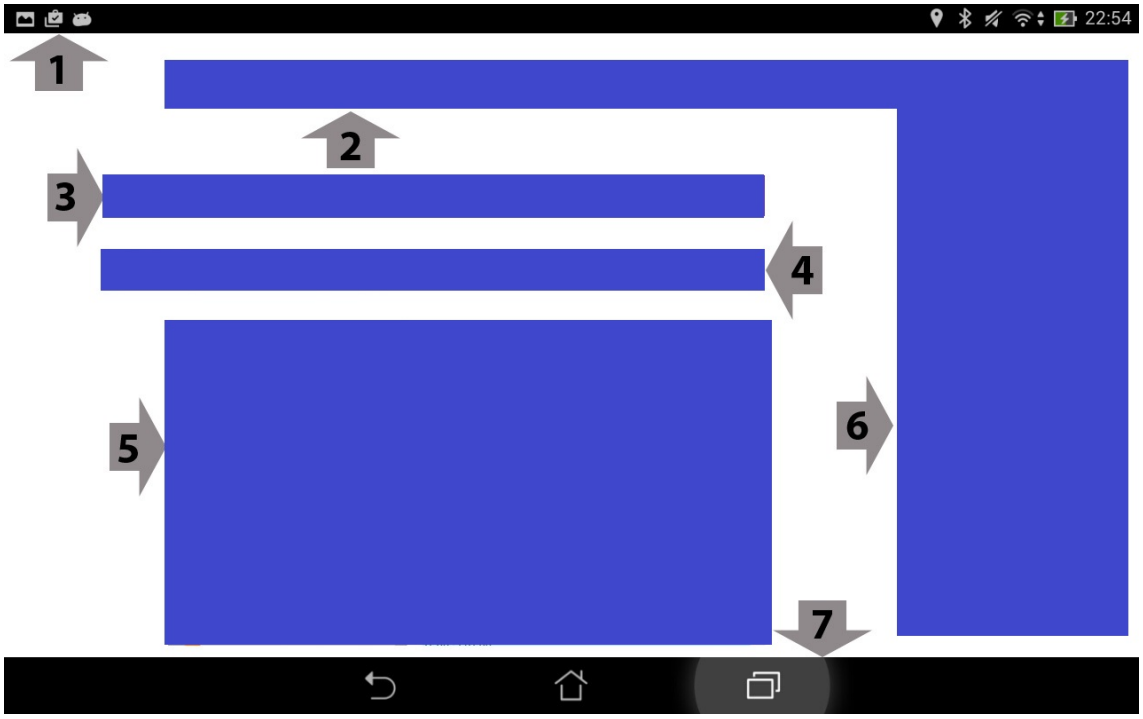
Obrázek 4: Trend zvětšování displejů.

2.4 Rozdělení obrazovky

V aplikaci myAvis NG mají všechny obrazovky velice podobné rozdělení, aby nedocházelo ke snížení přehlednosti. Pokud by měly obrazovky výrazně odlišné rozdělení, mohla by se aplikace stát méně intuitivní, komplikovanou, případně zcela neovladatelnou. Proto definujeme rozdělení, které je obecné pro všechny obrazovky. Rozdělení obrazovky je naznačené na obrázku 5. V obrázku jsou jednotlivé prvky, jako lišta filtrů, pracovní plocha apod. zmenšené a oddělené pro lepší znázornění. Ve výsledné aplikaci jsou samozřejmě jednotlivé prvky roztaženy tak, že do sebe zapadnou. Plná verze obrázku je dostupná pouze v neveřejné části diplomové práce na obrázku 3.

Při návrhu pracovní plochy v nové agendě musí software designer nebo analytik vycházet z grafických návrhů vzorových agend. Tímto způsobem je možné zachovat design a intuitivnost aplikace.

1. Stavový panel
2. Hlavní panel akcí
3. Sumarizační panel
4. Lišta filtrů
5. Pracovní plocha
6. Postranní panel
7. Panel s tlačítky Zpět, Domů a Nedávné



Obrázek 5: Rozdělení obrazovky.

2.4.1 Stavový panel

Klasický stavový panel operačního systému Android. V pravé části panelu jsou obvykle zobrazeny informace týkající se času, stavu baterie, reproduktoru, sítě apod. V levé části panelu najdeme informace týkající se služeb a aplikací, které běží na pozadí.

2.4.2 Hlavní panel akcí

Veškerá tlačítka se vyskytují v hlavním panelu akcí. V levé části panelu je titulek daného okna a tlačítko, které nás přesměruje do hlavního menu aplikace. V pravé části panelu jsou tlačítka, která mohou mít například funkci Uložit, Editovat, Spustit průvodce, Navigovat, Smazat apod. Na některých obrazovkách můžeme v hlavním panelu akcí nalézt také fulltextové vyhledávání, které je reprezentováno ikonou lupy. Při kliknutí na ikonu lupy se zobrazí editovatelné pole, do kterého můžeme zadat vyhledávaný řetězec. Používáme vlastní ikony, přizpůsobené Android prostředí. Na obrázku 4 neveřejné části jsou ukázány některé základní ikony.

2.4.3 Sumarizační panel

V sumarizačním panelu jsou různé informace týkající se dané obrazovky. Například v obrazovce s objednávkami a výdejkami je v sumarizačním panelu zobrazen součet cen vyfiltrovaných dokladů, zatímco v obrazovce se sklady je v sumarizačním panelu zobrazen ceník skladu a součet

jeho položek. Nejčastěji používaný ovládací prvek, který se v sumarizačním panelu používá, je obyčejný *TextView*. Při zadávání distribuce můžeme v sumarizačním panelu nalézt ovládací prvek *Spinner*. V některých případech je možné sumarizační panel schovat pomocí tlačítka umístěného v hlavním panelu akcí, například pokud chceme ušetřit místo pro seznam v pracovní ploše.

2.4.4 Lišta filtrů

Lištu filtrů používáme výhradně v obrazovkách, ve kterých je pracovní plocha tvořena seznamem položek. Použití filtru spolu s fulltextovým vyhledáváním, nacházejícím se v hlavním panelu akcí, samozřejmě omezuje množinu zobrazených položek. V liště filtrů používáme převážně ovládací prvky *ToggleButton*, *Spinner* a vytvořený *MultiSpinner*.

2.4.5 Pracovní plocha

Pracovní plocha slouží výhradně pro zobrazení a zadávání dat. Může se zde vyskytovat například seznam položek, seznam položek s detailem vybrané položky nebo pouhý formulář umožňující zakládání, editaci nebo zobrazení dat. Typické pracovní plochy si předvedeme v kapitole 2.6.

2.4.6 Postranní panel

Postranní panel dává uživateli užitečné informace a rady týkající se dané obrazovky. V některých případech postranní panel obsahuje vytvořený ovládací prvek *TileWidget* a následně funguje jako navigační panel, jak tomu je na obrázku 3 neveřejné části diplomové práce.

2.4.7 Panel s tlačítky Zpět, Domů a Nedávné

Jedná se o klasický panel s tlačítky operačního systému Android v softwarové nebo hardwarové podobě.

2.5 Hlavní menu

V hlavním menu aplikace myAvis NG se používá rozložení přístrojová deska, přeloženo z anglického slova „dashboard“, místo často používané vysunovací postranní navigace. Na následujícím obrázku 5 neveřejné části diplomové práce vidíme hlavní menu aplikace, ve kterém se používá již zmíněný ovládací prvek *TileWidget*. Jednotlivé lišty fungují jako tlačítka a navíc zobrazují užitečné informace, jako jsou počet aktivních zákazníků, počet nezahájených úkolů, stav pokladny, stav poslední synchronizace apod. Při dotyku lišty se otevře příslušná agenda. Můžeme si všimnou, že hlavní menu je podobné Modern UI - rozhraní vytvořeném firmou Microsoft.

Poznámka 1 Modern UI, také známé pod pojmem Metro, je označení uživatelského rozhraní vytvořeném firmou Microsoft. Modern UI se začalo používat v systémech Windows Phone a Windows 8. K jeho hlavním přednostem patří jednoduchost, zaměření na obsah a typografii.

2.6 Zobrazení

Pokud by měly obrazovky výrazně odlišné zobrazení, mohla by se aplikace stát méně intuitivní a komplikovanou. Existuje sada zobrazení, ze kterých se musí při návrhu nových oken vycházet. V této kapitole si představíme některá typická zobrazení.

2.6.1 Seznam

Seznam používáme především v případech, kdy nepotřebujeme zadávat data a stačí nám pouze jejich zobrazení. Pro zobrazení velkých datových sad využíváme ovládací prvky *RecyclerView*, *ListView* a *GridView*. Pomocí ovládacích prvků v liště filtrů a fulltextového vyhledávání můžeme omezit množinu zobrazených položek. Kliknutí na položku seznamu obvykle otevře obrazovku s detailem vybrané položky, zatímco dlouhé kliknutí otevře kontextové menu.

Barvy v seznamu jsou velice střídmé a občas se v položkách seznamu vyskytují ikony. Například v seznamu objednávek a výdejek se vyskytují různé ikony, které informují uživatele, zda je objednávka připravená, vyřízená nebo stornovaná. Často používaný prvek v seznamu je vertikální *TextView* s barevným pozadím, který slouží k rozlišení položek. Prvek je na položce seznamu umístěný úplně vlevo a vyskytuje se na většině seznamů v aplikaci. Implementace vertikálního *TextView* je přiložena k diplomové práci.

V aplikaci několik seznamů, které slouží hlavně ke sběru dat. Například obrazovka, ve které se provádí inventura skladu, má v pracovní ploše seznam, kde se v každé položce vyskytují ovládací prvky *EditText*, *Spinner*, *AutoCompleteTextView* a *CheckBox*. Používání editovatelných ovládacích prvků v položkách *ListView* se časem ukázalo velice nespolehlivé, vyplněná data se náhodně ztrácela. Řízení fokusu mezi editovatelnými ovládacími prvky se stává téměř nemožné. Rozhodl jsem se přidat do řešení *RecyclerView*, což je pokročilá, flexibilnější a rychlejší verze *ListView*, případně *GridView*. Implementace *RecyclerViewControl* bude předvedena v kapitole 5.2.1.

Na obrázku 6 neveřejné části diplomové práce je příklad seznamu zákazníků. V tomto konkrétním případě rozlišuje vertikální *TextView* s barevným pozadím jednotlivé typy zákazníků.

2.6.2 Seznam s detailem položky

Seznam s detailem položky používáme především v případech, kdy potřebujeme zadávat data nebo zobrazovat větší množství informací týkající se vybrané položky seznamu. Pomocí ovládacích prvků v liště filtrů a fulltextového vyhledávání můžeme omezit množinu zobrazených položek. Kliknutí na položku seznamu určuje informace zobrazené v detailu, zatímco dlouhé kliknutí otevře kontextové menu.

Barvy v seznamu jsou velice střídmé a občas se v položkách seznamu vyskytují ikony. Často používaný ovládací prvek v seznamu je *TextView* s kružnicovým pozadím, který například může informovat uživatele o skutečnosti, že změnil daný záznam.

Podporujeme také fragmenty v detailové části. Například v obrazovce s objednávkami a výdejkami jsou doklady v jednom seznamu. Při kliknutí na doklad se zobrazí buď fragment určený pro objednávku nebo fragment určený pro výdejku. V této obrazovce je detail položky přidán převážně z toho důvodu, že potřebujeme zobrazit položky daného dokladu.

Na obrázku 7 neveřejné části diplomové práce je zobrazen seznam sortimentu s detailem vybrané položky, kde je možné provést nacenění.

2.6.3 Detail

Detailové zobrazení používáme výhradně pro zadávání dat. Pokud se shora podíváme na rozložení pracovní plochy, vidíme hlavičku obsahující především ovládací prvky, jakou jsou *TextView* a *Spinner*. Následuje tělo formuláře, ve kterém zadáváme data pomocí různých ovládacích prvků, jako jsou *EditText*, *Spinner*, *AutoCompleteTextView* a *CheckBox*. V dolní části formuláře může být například seznam položek, příloh, fotografií a videí, graf, mapa apod.

2.6.4 Dialog

Pokud má dialog jednoduchou rozhodovací nebo potvrzovací funkci, pak stačí vytvořit obyčejný *AlertDialog*, který používáme například pro informování uživatele v případě, že špatně zadal data do formuláře, pro potvrzení před odstraněním nějakého záznamu apod.

Pokud má dialog komplikovanější funkci, pak je potřeba vytvořit v podstatě novou speciální obrazovku. Například v dialogu může být seznam tras a při výběru některé trasy se vygenerují návštěvy podle definovaného algoritmu.

2.7 Podpora různých zařízení

V rámci této práce vznikla podpora pro více typů rozlišení a velikostí obrazovky. Chování a vzhled aplikace by měl být stejný na všech zařízení. Android podporuje hustotně nezávislé jednotky, které mají označení *dp*. Pomocí těchto jednotek je možné nastavit velikost prvku uživatelského rozhraní relativně k hustotě pixelů konkrétního zařízení. Prvek uživatelského rozhraní s rozměry zadanými v *dp* získává velikost přepočtem *dp* na pixely podle hustoty pixelů konkrétního zařízení.

Třídy hustoty displeje jsou:

- ldpi (low),
- mdpi (medium),
- tvdpi (television),
- hdpi (high),
- xhdpi (extra-high),

- xxhdpi (extra-extra-high),
- xxxhdpi (extra-extra-extra-high).

Každá ze tříd hustoty displeje má svou konstantu, kterou zadané *dp* násobí a vypočítá tak skutečné pixely. Zadávání rozměrů pouze v těchto jednotkách však nestačí. Často nám pouhé násobení konstantou, kterou nemůžeme ovlivnit, nemusí vyhovovat.

Lze využít pojmenované rozměry, kde velikost definujeme zvlášť v souboru a následně se na rozměr pouze odkazujeme. Využitím pojmenovaných rozměrů můžeme mít více verzí těchto rozměrů v různých složkách zdrojů pro různá zařízení. Složky zdrojů pro různá zařízení se rozlišují pomocí sufixu, který specifikuje vlastnosti zařízení. Nejčastěji používanou vlastností je menší rozměr obrazovky, která se ve složkách definuje podle *dp* zařízení. Menší rozměr obrazovky budeme označovat zkratkou *sw*, z anglického výrazu „smallest width“.

Například obrázky ze složky *drawable-sw550dp* se použijí v případě, že zařízení má menší rozměr alespoň *550dp*. Pokud máme zařízení s rozlišením 1920×1200 a třídou hustoty *xhdpi*, pak se *dp* při přepočtu na pixely násobí dvěma. Menší rozměr obrazovky je *600dp* a tím pádem se použijí obrázky ze složky *drawable-sw550dp*.

Pokud začneme jakékoliv zdroje rozlišovat pomocí *sw*, má tento sufix přednost před ostatními. Přestanou nám fungovat zdroje rozdělené do složek pouze podle třídy hustoty displeje. Při nalezení složky pomocí *sw* se v případě jediné složky už neřeší další sufixy.

Například pokud by zdroje měly pouze složky *drawable-sw550-mdpi* a *drawable-xhdpi*, pak se pro tablet, který splňuje vlastnosti *sw550* a *xhdpi*, použijí zdroje ze složky *drawable-sw550-mdpi* místo zdrojů ze složky *drawable-xhdpi*.

Proto při používání menšího rozměru je nutné dělit složky určené pro obrázky navíc pomocí třídy hustoty displeje. Pro obrázky vzniká tolik variant, kolik je tříd rozdělení podle menšího rozměru násobeno počet podporovaných tříd hustoty displeje. Řešení myAvis NG obsahuje následující složky určené pro obrázky:

- drawable-sw1dp-mdpi, drawable-sw1dp-hdpi, drawable-sw1dp-tvdpi, drawable-sw1dp-xhdpi, drawable-sw1dp-xxhdpi,
- drawable-sw340dp-mdpi, drawable-sw340dp-hdpi, drawable-sw340dp-tvdpi, drawable-sw340dp-xhdpi, drawable-sw340dp-xxhdpi,
- drawable-sw550dp-mdpi, drawable-sw550dp-hdpi, drawable-sw550dp-tvdpi, drawable-sw550dp-xhdpi, drawable-sw550dp-xxhdpi,
- drawable-sw770dp-mdpi, drawable-sw770dp-hdpi, drawable-sw770dp-tvdpi, drawable-sw770dp-xhdpi, drawable-sw770dp-xxhdpi.

Jednotka *dp* sama řeší třídy hustoty displeje, a proto stačí mít stejné pojmenované rozměry pouze ve složkách, které mají sufix *sw*. Pro rozměry vzniká tolik variant, kolik je tříd rozdělení podle menšího rozměru. Řešení myAvis NG obsahuje následující složky určené pro rozměry:

- values-sw1dp,
- values-sw340dp,
- values-sw550dp,
- values-sw770dp.

Každá z těchto složek obsahuje XML soubor, ve kterém jsou vygenerované hodnoty rozměrů. Soubor je pojmenovaný *dimen-x*, kde *x* je koeficient, kterým se všechny rozměry násobí. Hodnoty se generují od *-50dp* do *150dp* s krokem *0.5* a od *150dp* do *1000dp* s krokem *1*. Hodnoty mají prefix *dp* pro kladné a prefix *dpMinus* pro záporné hodnoty. Například soubor *...values-sw1dp\dimen-0.65.xml* obsahuje následující vygenerované hodnoty:

```
<dimen name="dpMinus1.5">-0.98dp</dimen>
<dimen name="dpMinus1">-0.65dp</dimen>
<dimen name="dpMinus0.5">-0.33dp</dimen>
<dimen name="dp0">0dp</dimen>
<dimen name="dp0.5">0.33dp</dimen>
<dimen name="dp1">0.65dp</dimen>
```

Výpis 1: Vygenerované hodnoty rozměrů.

Je možné jednoduše přidat podporu pro další třídu menšího rozměru. Všechny rozměry v aplikaci by měly tyto hodnoty odkazovat následujícím způsobem:

```
<View
 android:layout_width="@dimen/dp50"
 android:layout_height="@dimen/dp100">
```

Výpis 2: Odkazování vygenerovaných hodnot rozměrů.

Implementoval jsem aplikaci, která generuje soubory *dimen-0.05.xml, dimen-0.1.xml, dimen-0.15.xml, ..., dimen-1.95.xml, dimen-2.xml*, které můžeme následně do Android řešení přidat.

Poznámka 2 Všechny *TextView* musí mít definovanou velikost textu. Pokud je použita výchozí velikost textu, nemůžeme ji nijak ovlivnit generovanými hodnotami rozměrů, proto by tato vlastnost *TextView* měla být vždy definována.

2.7.1 Škálování obrázků

Škálování obrázků je nutné v případě, že nemůžeme obrázku definovat rozměry. Například pokud je obrázek nastaven jako pozadí některého *View* nebo obrázkům, které jsou použité v hlavním panelu akcí. Škálování jednoho obrázku znamená v našem případě vytvoření 20 dalších obrázků. Vždy je důležité zvážit, zda nemůžeme nějakým způsobem obrázku definovat rozměry. Jednou z možností je obrázek obalit nějakým *View*, které má nastavenou velikost pomocí generovaných hodnot rozměrů.

Vytvořil jsem skript, který škáluje definovanou množinu obrázků do různých složek se zadaným poměrem. Skript pro škálování využívá EXE soubor vyvinutý třetí stranou. Jelikož se obrázky škálují a kopírují do velkého množství složek, vzniká mnoho kopií jednoho obrázku a obrázek se stává hůře udržitelný.

3 Možnosti implementace mobilních aplikací

Pokud vyvíjíme aplikaci, která má být používána na mobilních zařízeních, můžeme se vydat různými směry. Směr následně určuje, jak aplikace vzniká, které nástroje máme při vývoji k dispozici a vlastnosti, které daná aplikace může mít. V následujících kapitolách se pokusím tyto směry popsat.

3.1 Nativní aplikace

Obvyklý způsob vývoje aplikací je nativní vývoj, kdy je aplikace psána pro jednu konkrétní platformu. Každá platforma má svůj programovací jazyk. Pro zjednodušení nativního vývoje je často poskytována sada vývojových nástrojů a vývojové prostředí.

3.1.1 Výhody

Při nativním vývoji máme většinou přístup ke všem funkcím zařízení. Můžeme používat různé komponenty systému, jako jsou například fotoaparát, kalendář, mapu, senzory, pokročilé funkce zařízení, GPS apod. Můžeme získávat informace o stavu připojení, baterie, paměti, CPU apod. Mezi hlavní výhody nativní aplikace patří rychlost, spolehlivost a nízká spotřeba.

3.1.2 Nevýhody

Aplikace je vždy použitelná pouze na jedné platformě. Aplikaci vyvinutou pro operační systém Android není možné použít pro operační systém iOS apod. Tato skutečnost samozřejmě omezuje množinu potenciálních zákazníků. Dalším problémem nativního vývoje je nutnost znát konkrétní programovací jazyk, sadu vývojových nástrojů a vývojové prostředí pro danou platformu. Pokud má být aplikace určená pro více platforem, potřebujeme obvykle více pracovní síly a hardwarového vybavení.

Pokud vyváříme aplikaci, která má být rychlá, spolehlivá a úsporná je nativní vývoj ideální řešení.

3.2 Webová aplikace pro prohlížeč

Nativní vývoj může být v některých případech zbytečný a příliš drahý. Další způsob vývoje aplikací je vývoj webové aplikace pro prohlížeč. Aplikace se spouští ve webovém prohlížeči a je napsaná v HTML, který je často doplněný o CSS styly a JavaScript.

3.2.1 Výhody

Mezi hlavní výhody webové aplikace patří jednoduchá správa a instalace. Navíc je usnadněno nasazení nové verze, její konfigurace, aktualizace apod. Všichni uživatelé vždy používají aktuální verzi webové aplikace.

Jelikož aplikace běží na serveru, má jednodušší přístup k databázi a dalším zdrojům. Pro tvorbu uživatelského rozhraní v HTML je možné použít téměř jakoukoliv platformu, protože většina dnešních mobilních zařízení má webový prohlížeč. Webová aplikace je tedy přístupná uživatelům, kteří disponují takzvaným chytrým zařízením.

3.2.2 Nevýhody

Aplikace se spouští ve webovém prohlížeči, na kterém je díky tomu závislá. Volba prohlížeče může značně ovlivnit vzhled a funkčnost aplikace. Uživatelské rozhraní je renderováno v prohlížeči a jakákoliv chyba ovlivní vzhled nebo v horším případě funkčnost aplikace. Existuje spousta webových prohlížečů pro různé operační systémy a není možné ovlivnit, který webový prohlížeč uživatel použije.

Při používání webové aplikace je potřeba připojení k Internetu, což může být pro řadu aplikací velký problém. Často se totiž stává, že se uživatel pohybuje v oblasti, kde je připojení k Internetu slabé nebo zcela nemožné. Následně se aplikace stává pomalou, případně zcela nepoužitelnou.

3.2.3 Prohlížeče pro Android

V současnosti je standardní systémový prohlížeč Chrome. Ačkoliv dávají mnozí vývojáři aplikací pro Android přednost vestavěnému prohlížeči, k dispozici jsou i jiné prohlížeče. Zde je uvedeno několik nejznámějších alternativ, které podporují standard HTML5:

- Firefox,
- Opera,
- Dolphin.

3.2.4 JavaScript

JavaScript je multiplatformní, objektově orientovaný skriptovací jazyk. Používá se jako interpretovaný programovací jazyk pro webové stránky. Interpretace JavaScriptu je pomalá při srovnání s kompilovaným nativním kódem, přesto se výkonnost JavaScriptu stále zvyšuje. Přibývající JavaScriptové frameworky, knihovny a funkce značně usnadňují vývoj mobilních aplikací napsaných v HTML. Frameworky jsou nejčastěji určeny k vývoji uživatelského rozhraní, řízení grafických elementů a řízení interakce dotykového displeje. Tyto frameworky, knihovny a funkce navíc sjednocují rozhraní různých mobilních prohlížečů. Zde je uvedeno několik nejznámějších frameworků a knihoven:

- jQuery Mobile,
- Sencha Touch,

- Cocos2D,
- ZeptoJS,
- Impact.js,
- DHTMLX Touch.

3.3 Multiplatformní nativní aplikace

Při nativním vývoji aplikace máme možnost zapojit různé multiplatformní prostředky a přístupy. Snažíme se, aby maximální množství vývojářské práce bylo odvedeno pouze jednou a následná implementace pro různé platformy byla co nejsnazší. V následujících kapitolách se seznámíme s multiplatformními prostředky a přístupy, které se při tvorbě multiplatformní nativní aplikace využívají.

3.3.1 Obalení kódu

Obalení kódu využívá toho, že nativní aplikace umožňují vytvořit komponentu, která umožnuje zobrazovat webové stránky. Například u platformy Android se využívá komponenta *WebView*. Programátor následně aplikaci implementuje pomocí HTML, CSS a JavaScriptu. Mohou samozřejmě existovat obalení i pro další jazyky.

Nativní aplikace spustí komponentu, která umožňuje zobrazovat webové stránky a v této komponentě je zobrazena aplikace implementována pomocí HTML, CSS a JavaScriptu. Takovým způsobem vzniká webová aplikace, která nepotřebuje webový prohlížeč.

3.3.2 Překladač

Překládá vysokoúrovňový zdrojový kód do mezikódu, nativního jazyka nebo do nízkoúrovňového strojového kódu. Překladač kontroluje různé limity, rozsah a chyby ve zdrojovém kódu. Musí procházet celý zdrojový kód, proto překlad může trvat déle. Zde je uvedeno několik nejznámějších překladačů:

- Xamarin,
- Marmelade,
- MoSync.

Při implementaci multiplatformní nativní aplikace pomocí nástroje Xamarin se využívá následující princip. Aplikační logika je implementována pouze jednou v jazyce C Sharp. Následně pro různé platformy implementujeme pouze uživatelské rozhraní.

Uživatelské rozhraní se u různých platforem většinou výrazně liší, zatímco aplikační logika je pro všechny platformy společná. Obecně můžeme říci, že aplikační logika se implementuje v nějakém rozšířeném programovacím jazyce.

3.3.3 Runtime

Runtime je obecný pojem, který se odkazuje na nějakou knihovnu, framework nebo platformu, na které běží námi implementovaný kód. Runtime popisuje software a instrukce, které se vykonávají za běhu programu. Zvláště popisuje ty instrukce, které nebyly napsány explicitně a jsou nezbytné pro správné vykonávání kódu.

Jedná se o knihovny založené na některém z programovacích jazyků. Tyto knihovny existují pro různé platformy v různých variantách a rozhraní těchto variant je podobné, ideálně stejné. Runtime je multiplatformní vrstva, která odstiňuje aplikaci od rozdílů různých platforem.

Nízkoúrovňové jazyky, jako například C, mají velice malý runtime. Více komplexní jazyky, jako například Objective-C, Java a C Sharp, mají mnohem větší runtime. Zde jsou uvedeny příklady runtime:

- Unity,
- Adobe Flex,
- Corona.

3.4 PhoneGap

PhoneGap je pravděpodobně úplně první alternativní aplikační prostředí pro Android, které bylo uvedeno na trh již na začátku roku 2009. Projekt PhoneGap je open-source a stojí za ním společnost Nitobi, poskytovatel kombinace produktů open-source, komerčních produktů a poradenských a výukových služeb. V říjnu roku 2011 byla společnost Nitobi převzata společností Adobe, a aby zajistila dlouhou životnost zdrojového kódu prostředí PhoneGap, předala jej organizaci Apache Software Foundation, kde byl projekt přejmenován na Apache Callback.

PhoneGap je platforma založená na standardu HTML5, která umožňuje vývoj aplikací na základě jednotné technologie a jejich nasazení na více různých platformách. Práce s prostředím je jednoduchá a sestává z následujících kroků:

- vytvoření aplikace pomocí standardních webových programovacích jazyků, například HTML5 nebo JavaScript,
- obalení aplikace obálkou poskytovanou prostředím PhoneGap, což umožňuje přístup k potřebným rozhraním API,
- nasazení aplikace na různých platformách.

Aplikace založená na prostředí PhoneGap je tvořena kódem jazyků HTML, CSS a JavaScript. Neliší se od mobilního webu nebo aplikace naprogramované v prostředí HTML5, až na to, že v prostředí PhoneGap jsou webové prostředky přibaleny k aplikaci a nestahují se při jejím spuštění.

Aplikace naprogramované v prostředí PhoneGap vypadají spíše jako webové stránky než jako nativní aplikace pro Android. Webová aplikace nikdy nebude vypadat a fungovat jako nativní aplikace, přesto je prostředí PhoneGap často používáno pro vývoj aplikací.

3.5 Shrnutí

V předešlých kapitolách jsem popsal různé přístupy k tvorbě aplikací pro mobilní zařízení. Popsal jsem vývoj nativní aplikace, který je sice nákladný, ale software je kvalitní a optimalizovaný. Následně jsem popsal vývoj webové aplikace, který je snadný a rychlý, bohužel rychlost a možnosti softwaru jsou značně omezeny. Na závěr jsem popsal smíšený vývoj aplikace, kde jsem se zaměřil na využití multiplatformních prostředků, jako jsou obalení kódu, překladač a runtime.

Každá z těchto možností vývoje mobilních aplikací má své výhody a nevýhody. Rozhodnutí, kterou cestou se při tvorbě aplikace vydáme, je zcela zásadní. Společnost Kvados se vydala svou vlastní cestou, kterou si popíšeme v následující kapitole.

4 AndroidClient

Vývoj aplikací ve společnosti Kvados vychází z principů a tříd, které jsou používány pro vývoj WinForm SmartClient aplikací. Pojem AndroidClient používáme pro označení mobilní klientské aplikace, která je určena pro platformu Android. Chování aplikace se popisuje pomocí metadat, která jsou zapsána do XML souborů. Struktura metadat pro AndroidClient aplikaci je podobná struktuře metadat pro WinForm SmactClient aplikaci. Hlavní rozdíl spočívá v odlišnosti systémových typů, které jsou implementovány v programovacím jazyce Java. Téměř všechny produkty společnosti Kvados využívají metadata:

- AndroidClient má systémové typy implementovány v jazyce Java,
- WebClient má systémové typy implementovány v jazyce TypeScript,
- SmartClient má systémové typy implementovány v jazyce C Sharp.

Metadata pomáhají vývojářům velice snadno vytvářet nové agendy a procesy. Jedná se o dlouhodobě používaný a prověřený způsob vývoje.

Metadata jsou zároveň multiplatformní vrstva, kterou můžeme využít v případě, že bychom chtěli naše řešení převést například na platformu iOS. Museli bychom přepsat stávající systémové typy, které jsou implementovány v programovacím jazyce Java, do programovacího jazyka Objective-C.

Používání metadat je výhodné i v případě, že chceme přeškolit programátora na jiný produkt. Pokud programátor zná způsob vývoje AndroidClient aplikace, může být jednoduše přeškolen například na vývoj WebClient aplikace.

Pro implementaci systémových typů používáme prostředí Android Studio. Pro definování a sestavení metadat používáme prostředí Visual Studio, ve kterém současně pracujeme s TFS repozitářem.

Instalace Visual Studia je rozšířena o možnost sestavení metadat. Při sestavení metadat se kontrolují systémové typy. Pokud není možné systémový typ v řešení nalézt, sestavení metadat skončí s chybou. Sestavená metadata se kopírují do složky *assets* aplikačního modulu a jsou součástí výsledného APK.

V této kapitole představím pojem softwarová továrna a její konkrétní implementaci Smart Client Software Factory, ze které vychází vývoj aplikací ve společnosti Kvados. Dále předvedu strukturu a definici objektu *WorkItem*. Tyto informace využijeme v následující kapitole 5, ve které si probereme implementaci komponent uživatelského rozhraní pro AndroidClient aplikaci.

4.1 Softwarová továrna

Jedná se o kolekci softwarových prostředků, které jsou nainstalovány ve vývojovém prostředí. Prostředky pomáhají architektům a vývojářům tvořit dobře modifikovatelné aplikace. Pomocí softwarové továrny se vyvíjí aplikace, které mají jednotnou architekturu a rysy.

Důležitými prvky softwarových továren jsou prostředky, která zahrnují opětovně použitelný kód, dokumentaci apod. Dalšími součástmi mohou být nástroje generující kód, průvodci a návrhy grafického uživatelského rozhraní. Továrna poskytuje šablony a nástroje, pomocí nichž se urychluje vytváření nové aplikace.

4.2 Smart Client Software Factory

SCSF poskytuje integrovanou sadu řízení, která pomáhá vývojářům velice jednoduše vytvářet složité aplikace. Ty mohou mít některé charakteristické rysy, jako jsou například bohaté uživatelské rozhraní, snadné nasazení, jednoduchá konfigurace, sdílení dat s jinými systémy a podobné užitečné rysy. Dalším typickým znakem je používání lokálního uložení dat a jejich následné zpracování například po výpadku sítě. SCSF umožňuje architektům a vývojářům vyvíjet aplikace složené z modulů, které mohou být vyvíjeny, sestavovány a distribuovány nezávislými týmy.

4.3 WorkItem

Základní prvek AndroidClient aplikace je *WorkItem*, který vytváří především instanci aktivity nebo fragmentu. Objekt *WorkItem* shrnuje logiku pro jeden případ užití a udržuje stav pro všechny případy či podpřípady užití. AndroidClient aplikace se skládá z modulů, ve kterých jsou *WorkItemy* umístěny. *WorkItem* je v metadatech reprezentován XML souborem.

Funkčnost *WorkItemu* je rozšiřována pomocí extenzí a data jsou uchovávána v *ControlDatech*. Uživatelské rozhraní je definováno pomocí sekce *View*, která se skládá z *Panelů*, *ViewControlů* a *ControlItemů*. Jednotlivé *WorkItemy* si mohou předávat data pomocí relací a konektorů.

Každý objekt ve *WorkItemu*, který implementuje rozhraní *IParametrizable*, může mít v metadatech definovány vlastní parametry. Při inicializaci objektu dochází k načtení definovaných parametrů z metadat do klíčované kolekce, odkud můžeme získat hodnoty jednotlivých parametrů. Metody, které vracejí hodnoty jednotlivých parametrů, jsou v parametrizovaném objektu uvedeny pomocí anotací. Následující kód 3 ukazuje strukturu a prázdnou definici *WorkItemu*.

```
<ModuleSchemaPart xmlns="http://schemas.kvados.com/2010/07/QAS/SmartClient/
    ModuleSchema">
 <WorkItems>
  <WorkItem Key="Menu">
   <InConnectors></InConnectors>
   <OutConnectors></OutConnectors>
   <BuilderStrategies></BuilderStrategies>
   <Services></Services>
   <Extensions></Extensions>
   <Views></Views>
   <DataContainer></DataContainer>
   <ToolItems></ToolItems>
```

```
   <Commands></Commands>
  </WorkItem>
 </WorkItems>
</ModuleSchemaPart>
```

Výpis 3: Struktura a prázdná definice *WorkItemu*.

4.3.1 RootWorkItem

RootWorkItem se spouští automaticky při startu AndroidClient aplikace. Každý modul může mít vlastní *RootWorkItem*, které se při spuštění AndroidClient aplikace sloučí do jednoho. Instance *RootWorkItemu* je dostupná přes singleton na objektu *ACApp*.

```
WorkItem root = ACApp.getCurrent().getRootWorkItem();
```

Výpis 4: Získání instance *RootWorkItemu* přes singleton na objektu *ACApp*.

Další *WorkItemy* se z *RootWorkItemu* spouštějí pomocí extenze. Následující příklad 5 ukazuje spuštění *WorkItemu*, který má klíč *UpdateService* a nachází se v modulu *QAS.AndroidClient.Core*.

```
<Extension Type="kvados.qas.androidclient.core.extensions.
    ExecuteWorkItemWIExtension">
 <Parameters>
  <Parameter Key="WorkItem" Value="UpdateService,QAS.AndroidClient.Core"
     ValueMode="WorkItemRef"/>
 </Parameters>
 <Triggers>
  <WorkItemEventTrigger EventTriggerType="RunStarted"/>
 </Triggers>
</Extension>
```

Výpis 5: Spuštění *WorkItemu* pomocí extenze.

4.3.2 Konektory

Objekt *WorkItem* obsahuje kolekci vstupních a výstupních konektorů. Vstupní konektory slouží pro předávání dat do *WorkItemu* a výstupní konektory slouží pro předávání dat z *WorkItemu*. Data se předávají pomocí relací, které jsou definovány mimo definici *WorkItemu*. Relace spojí výstupní konektor *WorkItemu* rodiče s vstupním konektorem *WorkItemu* potomka. Následující příklad 6 ukazuje jeden vstupní a jeden výstupní konektor.

```
<InConnectors>
 <Connector Key="TaskUUID" ValueType="java.lang.UUID" ValueMode="Single" Usage=
    "Required"/>
```

```
</InConnectors>
<OutConnectors>
 <Connector Key="Customer" ValueType="kvados.crp.myavis.customers.entitymodel.
    MblCustomersEntity" ValueMode="Single" Usage="Required"/>
</OutConnectors>
```

Výpis 6: Vstupní a výstupní konektor.

4.3.3 Relace

Existují dva typy relací, přímá *DirectBindingRelation* a stavová *StateRelation* relace. Přímá *DirectBindingRelation* slouží pro propojení výstupního konektoru *WorkItemu* rodiče s vstupním konektorem *WorkItemu* potomka. Stavová *StateRelation* propojuje stav *WorkItemu* rodiče s vstupním konektorem *WorkItemu* potomka. Stavová relace má výhodu v tom, že není potřeba specifikovat klíč *WorkItemu* rodiče.

Stav objektu *WorkItem* je kolekce klíčovaných objektů. Následující příklad 7 ukazuje vkládání objektu do stavu *WorkItemu* a získání objektu ze stavu *RootWorkItemu*.

```
getWorkItem().getState().set(key,value);
getWorkItem().getRootWorkItem().getState().get(key);
```

Výpis 7: Vložení a získání objektu ze stavu *WorkItemu*.

4.3.4 BuilderStrategy

Do AndroidClient aplikace lze zaregistrovat objekty, kterým se říká *BuilderStrategy*. Objekt *BuilderStrategy* má dvě metody *BuildUp* a *TearDown*. Pokud přidávám objekt do kolekce na *WorkItemu*, vyvolá se na všech zaregistrovaných *BuilderStrategy* metoda *BuildUp* s parametrem, který obsahuje přidávaný objekt. Pokud naopak odebírám objekt z kolekce na *WorkItemu*, vyvolá se na všech zaregistrovaných *BuilderStratery* metoda *TearDown* s parametrem, který obsahuje odebíraný objekt. Takovým způsobem lze ovlivnit vytváření různých objektů. Tyto strategie slouží například k vytváření hlavního panelu akcí z *ToolItemů*. Registrace kolekce *BuilderStrategies* se provádí nejčastěji v *RootWorkItemu*. Níže uvedený příklad 8 uvádí registraci kolekce *BuilderStrategies*:

- *ToolbarsStrategy* slouží k vytváření hlavního panelu akcí z *ToolItemů*,
- *ContextMenuStrategy* slouží k vytváření kontextového menu z *ToolItemů*,
- *ButtonCommandExecuteStrategy* slouží k propojení tlačítka na detailu formuláře s *Commandem*.

```
<BuilderStrategies>
 <BuilderStrategy Key="ToolbarsStrategy" Type="kvados.qas.androidclient.core.
    builderstrategies.ToolbarsStrategy"/>
 <BuilderStrategy Key="ContextMenuStrategy" Type="kvados.qas.androidclient.core
    .builderstrategies.SystemContextMenuStrategy"/>
 <BuilderStrategy Key="ButtonCommandExecuteStrategy" Type="kvados.qas.
    androidclient.core.builderstrategies.ButtonCommandExecuteStrategy"/>
</BuilderStrategies>
```

Výpis 8: Registraci kolekce *BuilderStrategies*.

4.3.5 Služby

Služby zapouzdřují funkčnost běžnou pro celou aplikaci nebo jen pro objekt *WorkItem*. Mezi typické služby se řadí například bezpečnostní služby zodpovědné za autentifikaci či autorizaci uživatele. Dále existují služby pro výpočet ceny dokladu, zaokrouhlení dokladu, přepočet množství mezi měrnými jednotkami apod. Řadu takových služeb jsem pro aplikaci myAvis NG sám vytvořil.

Jakákoliv třída, která implementuje libovolné rozhraní, může být zaregistrována jako služba. Službu můžeme zaregistrovat na libovolný *WorkItem*, obvykle však službu registrujeme na *RootWorkItem*, aby byla dostupná v celé aplikaci. Vytvoření instance služby se provádí při vytvoření instance příslušného *WorkItemu*, na který je služba registrována. Klíč při registrování služby je rozhraní a typ je třída, která rozhraní implementuje. Registrování služby je uvedeno na následujícím příkladu 9.

```
<Service Key="kvados.qas.androidclient.core.errorhandlers.IErrorHandlerService"
    Type="kvados.qas.androidclient.core.errorhandlers.ErrorHandlerService"/>
```

Výpis 9: Registrování služby.

Přístup ke službě je možný z instance *WorkItemu*. Získání služby je uvedeno na následujícím příkladu 10.

```
getWorkItem().get(IErrorHandlerService.class);
```

Výpis 10: Získání služby.

4.3.6 Command

Command představuje akci, kterou daný *WorkItem* umožňuje vykonat. *Command* může mít definovaný typ. Pokud typ není uveden, pak se vytváří výchozí *Command*, který slouží především k řízení extenzí. V následujícím příkladu 11 je definován *Command*, který při vyvolání spustí další *WorkItem*.

```
<Command Key="WorkTasks" Type="kvados.qas.androidclient.core.commands.
    ExecuteWICommand">
 <Parameters>
  <Parameter Key="WorkItem" Value="WorkTasks" ValueMode="WorkItemRef"/>
 </Parameters>
</Command>
```

Výpis 11: *Command* spouštějící další *WorkItem*.

4.3.7 Extenze

Extenze obsahují hlavní funkčnost objektu *WorkItem*. Extenze je definována typem, parametry a spouštěmi. Každá extenze musí rozšiřovat třídu *WorkItemExtension*, ve které dochází k registrování na různé typy spouští. K vykonání extenze dojde v případě, že se provede některá spoušť. V extenzích můžeme například generovat objednávky, kontrolovat zadané data, mazat záznamy z databáze apod. V následujícím příkladu 12 je extenze, která při spustění *WorkItemu* vykoná *Command*.

```
<Extension Type="kvados.qas.androidclient.core.extensions.commands.
    ExecuteCommandWIExtension">
 <Parameters>
  <Parameter Key="Command" Value="Edit" ValueMode="Command"/>
 </Parameters>
 <Triggers>
  <WorkItemEventTrigger EventTriggerType="RunStarted"/>
 </Triggers>
</Extension>
```

Výpis 12: Extenze spouštějící *Command*.

Existují následující typy spouští:

- *UIComponentEventTrigger* se vyvolá při vzniku události na komponentě uživatelského rozhraní,
- *CommandTrigger* se vyvolá při vzniku události na *Commandu*,
- *ControlDataEventTrigger* se vyvolá při vzniku události na *ControlDatech*,
- *ServiceEventTrigger* se vyvolá při vzniku události ve službě,
- *WorkItemEventTrigger* se vyvolá při vzniku události na *WorkItemu*.

4.3.8 ToolItems

Slouží k vytváření například hlavního panelu akcí, kontextového menu, propojení tlačítka s *Commandem* apod.

```
<ToolItems>
  <Toolbar Key="Menu" Strategy="ToolbarsStrategy">
    <ToolbarItem Key="New" TextRef="New">
      <Parameters>
        <Parameter Key="ShowAsAction" Value="Never" ValueMode="Object"
            ValueType="java.lang.String" />
      </Parameters>
      <CommandExecute CommandName="New"/>
    </ToolbarItem>
  </Toolbar>
</ToolItems>
```

Výpis 13: Definice kontextového menu.

4.3.9 View

View je zodpovědné za prezentování části nebo celého modelu uživateli. Objekt *View* popisuje uživatelské rozhraní a umožňuje modifikaci obsahu pomocí uživatelských ovládacích prvků. Při definování *View* existují následující pravidla:

1. *View* může obsahovat kolekci *Panelů*,
2. *Panel* může obsahovat kolekci *Panelů* a kolekci *ViewControlů*,
3. *ViewControl* může obsahovat kolekci *ControlItemů*,
4. *ControlItem* může obsahovat kolekci *ControlItemů*.

Každá z těchto komponent uživatelského rozhraní je definovaná svým typem a parametry. Velká část implementace této diplomové práce se týká právě vytváření nových *ViewControlů* a *ControlItemů*. Následující příklad 14 ukazuje zkrácenou definici *View*.

```
<Views>
 <View Key="View" Type="kvados.qas.androidclient.core.ui.LinearLayoutView">
  <Parameters/>
  <WorkspaceInfo Workspace="MainWorkspace"/>
  <ViewPanels>
   <ViewPanel Key="Panel" Type="kvados.qas.androidclient.core.ui.
       VirtualViewPanel">
```

```
    <ViewControls>
     <ViewControl Key="LoggedUser" Type="kvados.qas.androidclient.core.ui.
         detail.SimpleDetailViewControl" ControlData="LoggedUser">
      <Parameters>
       <Parameter Key="Layout" Value="layout/myfaber_signed_user_layout"
           ValueMode="Object" ValueType="java.lang.String"/>
      </Parameters>
      <ControlItems>
       <ControlItem Key="id/myfaber_signed_user_name">
        <Parameters>
         <Parameter Key="DataProperty" Value="FullName" ValueMode="Object"
             ValueType="java.lang.String"/>
        </Parameters>
       </ControlItem>
      </ControlItems>
     </ViewControl>
    </ViewControls>
   </ViewPanel>
  </ViewPanels>
 </View>
</Views>
```

Výpis 14: Zkrácená definice *View*.

4.3.10 Entitní model

V AndroidClient aplikaci používáme entitní model, který umožnuje pracovat s databází pomocí tříd, které reprezentují jednotlivé tabulky databáze. Třída reprezentující záznam některé tabulky se nazývá entita. Instance dané třídy následně reprezentuje konkrétní záznam v dané tabulce. Entity jsou generovány automaticky z databáze.

4.3.11 ControlData

Definovaná kolekce objektů typu *ControlData* se vytvoří při vytvoření instance příslušného *WorkItemu*. Následně můžeme do *ControlData* vkládat různé objekty, se kterými potřebujeme pracovat v rámci daného *WorkItemu*. Objekt *ControlData* umožňuje držet libovolný objekt nebo pole objektů, které můžeme používat v extenzích, vkládat do konektorů a vázat do *ViewControlů*. Více o vázání do *ViewControlů* je uvedeno v jedné z následujících kapitol 4.3.12. Níže uvedený příklad 15 ukazuje definici kolekce objektů typu *ControlData*.

```
<DataContainer>
```

```
 <ControlData Key="Customer" Type="kvados.qas.androidclient.core.data.
    SingleValueControlData"/>
 <ControlData Key="Customers" Type="kvados.qas.androidclient.core.data.
    SingleListControlData"/>
</DataContainer>
```

Výpis 15: Definice kolekce objektů typu *ControlData*.

Zde jsou vypsány základní typy *ControlData*:

- *SingleValueControlData* drží objekt,
- *SingleListControlData* drží pole objektů,
- *PersistedValueControlData* drží entitu,
- *PersistedListControlData* drží pole entit.

4.3.12 Vázání

Vázání, přeloženo z anglického slova „binding“, je nedílnou součástí všech Android aplikací. Jedná se o nastavení vlastností ovládacího prvku za běhu aplikace, jinak řečeno vázání je nastavení vlastností ovládacího prvku skrz aplikační logiku.

V AndroidClient aplikaci se pro definici vázání ovládacích prvku využívá kolekce objektů typu *ControlItem*, které svými parametry určují vlastnosti ovládacích prvků za běhu aplikace. Obyčejný *ControlItem* může mít parametry, jako jsou *DataProperty*, *VisibleProperty* a *EnabledProperty*. Hodnota těchto parametrů je identifikátor vlastnosti vázaného objektu. Z vázaného objektu získáme pomocí reflexe hodnotu dané vlastnosti, kterou následně vážeme do ovládacího prvku. Vázání probíhá v pomocné metodě, které předáme vázaný objekt, instanci ovládacího prvku a *ControlItemu*.

- *DataProperty* - Hodnota vlastnosti vázaného objektu určuje zobrazený text v případě použití prvku *TextView*, *EditText* a *AutoCompleteTextView*. V případě použití prvku *Spinneru* určuje vybranou položku.
- *VisibleProperty* - Hodnota vlastnosti vázaného objektu určuje, jestli je ovládací prvek viditelný.
- *EnabledProperty* - Hodnota vlastnosti vázaného objektu určuje, jestli je možné ovládací prvek upravovat.
- *ImmediateBinding* - V případě použití *EditText* a *AutoCompleteTextView* určuje, zda se projeví vázání ihned při změně textu nebo až při ztrátě fokusu.

Pokud ovládací prvek vyžaduje speciální parametry, musí implementovat rozhraní *IViewControlItemControl*. Následně má ovládací prvek dostupnou instanci *ControlItemu* i s jeho parametry, které může využít pro vlastní účely. Například ovládací prvek *DataQueryAutoCompleteTextView* implementuje výše uvedené rozhraní, protože vyžaduje speciální parametry. Tento ovládací prvek si více přiblížíme v kapitole 5.1.2.

ViewControl je určen svým typem a při své inicializaci vytváří Android ovládací prvek nebo vázané rozvržení. Pro vázané rozvržení musí platit, že obsahuje alespoň tolik ovládacích prvků, kolik je *ControlItemů*. Navíc pro každý *ControlItem* musíme v rozvržení uživatelského rozhraní najít ovládací prvek, který má shodné *android:id* s klíčem *ControlItemu*.

ViewControl vytváří a udržuje instanci třídy, která implementuje rozhraní *IBindingSource*. Tato instance pomáhá při vázání vlastností objektů do ovládacích prvků. V AndroidClient aplikaci se setkáváme se dvěma způsoby vázání vlastností objektu do ovládacích prvků pomocí *ControlItemů*.

- *ControlData* obsahují pole objektů. Typ *ViewControlu* musí být takový, aby vytvořil ovládací prvek, který umožňuje zobrazovat velké datové sady. Ovládacímu prvku obvykle vytvoříme nějaký adaptér, který se stará o vytváření vázaných rozvržení uživatelského rozhraní pro objekty nacházející se v *ControlData*. *IBindingSource* se stará o naplnění adaptéru, zatímco adaptér se stará o vázání vlastností objektů do ovládacích prvků ve vytvořených rozvrženích.

- *ControlData* obsahují jeden objekt. Typ *ViewControlu* musí být takový, aby vytvořil vázané rozvržení uživatelského rozhraní. *IBindingSource* se stará o vázání vlastností objektu do ovládacích prvků ve vytvořeném rozvržení.

Příklad 1
ListViewControl vytvoří ovládací prvek *ListView* a adaptér, který se stará o vytvoření vázaných rozvržení pro pole objektů nacházejících se v *ControlData*. *ListBindingSource* se stará o naplnění adaptéru, zatímco adaptér se stará o vázání vlastností objektů do ovládacích prvků ve vytvořených vázaných rozvrženích. ■

Příklad 2
GridViewControl vytvoří ovládací prvek *GridView* a adaptér, který se stará o vytvoření vázaných rozvržení pro pole objektů nacházejících se v *ControlData*. Stejně jako u *ListViewControl* se vytváří instance *ListBindingSource*, která se stará o naplnění adaptéru. Adaptér se stará o vázání vlastností objektů do ovládacích prvků ve vytvořených vázaných rozvrženích. ■

Příklad 3
SimpleDetailViewControl vytvoří vázané rozvržení pro objekt nacházející se v *ControlData*. *DetailBindingSource* se stará o vázání vlastností objektu do ovládacích prvků ve vytvořeném rozvržení. ■

Rozšiřujících tříd *ViewControlu* existuje málo, jedná se o obecné komponenty, které mohou být použity na více místech. Implementaci *RecyclerViewControl* a *DataQueryAutocompleteTextView* si předvedeme v následující kapitole 5.

5 Implementace komponent uživatelského rozhraní

V předchozí kapitole jsem nastínil způsob vytváření AndroidClient aplikace. Představil jsem objekt *WorkItem*, ve kterém existuje sekce *View* skládající se z *ViewPanelů*, *ViewControlů* a *ControlItemů*. Objekt *ViewControl* vytváří a řídí buď konkrétní Android prvek nebo rozvržení uživatelského rozhraní a současně řídí interakci mezi *WorkItemem* a Android prvkem nebo rozvržením uživatelského rozhraní.

Příklad 4
ListViewControl vytváří ovládací prvek *ListView* a kliknutí na položku seznamu oznamuje *WorkItemu*, ve kterém může být extenze zaregistrovaná právě na tuto událost. ■

Příklad 5
RecyclerViewControl vytváří ovládací prvek *RecyclerView* a při přetažení položky na jinou pozici spustí určený *Command* ve *WorkItemu*, který nese informace o přetažení položky. Na vykonání *Commandu* může reagovat extenze, ve které máme nyní dostupnou informaci o původní a nové pozici vybrané položky. Extenze může například upravit data v databázi apod. ■

Při vývoji aplikace se setkáváme s následujícími způsoby implementace komponent uživatelského rozhraní.

- Vlastní komponenta - Vytvoříme podtřídu vybraného Android ovládacího prvku nebo rozvržení uživatelského rozhraní a přepíšeme vybrané metody.
- *ViewControl* komponenta - Vytvoříme podtřídu *ViewControl*, ve kterém pracujeme s Android prvkem nebo rozvržením.
- *ControlItem* komponenta - Vytvoříme podtřídu vybraného Android ovládacího prvku a navíc implementujeme rozhraní *IViewControlItemControl*. Při inicializaci *View* daného *WorkItemu* se zavolá metoda rozhraní *initialize(IViewControlItem)*, ve které získá ovládací prvek instanci *ContolItemu*.

Při vývoji aplikace se obvykle vytváří nové *ViewControly*, které používají původní Android ovládací prvky, jako jsou *ListView*, *GridView*, *RecyclerView*, *Spinner* apod. Méně často se vytváří zcela nové prvky, pro které většinou *ViewControl* dopisujeme, aby ovládací prvek mohl komunikovat s *WorkItemem*.

Rozšiřujících tříd *ViewControlu* existuje málo, jedná se o obecné komponenty, které mohou být použity na více místech. Mezi největší výhody *ViewControl* a *ControlItem* komponent patří jejich znovupoužitelnost. Například při tvorbě nového seznamového zobrazení stačí vytvořit rozvržení uživatelského rozhraní a správně definovat metadata, zatímco do zdrojových kódu napsaných v Javě nemusíme teoreticky vůbec sahat. V následujících kapitolách si předvedeme implementaci komponent uživatelského rozhraní výše uvedenými způsoby.

5.1 Implementace vlastních komponent

Implementace vlastních komponent se provádí tak, že vytvoříme podtřídu vybraného ovládacího prvku nebo rozvržení uživatelského rozhraní a přepíšeme vybrané metody.

Jestliže se chystáme mírně změnit některý z předpřipravených ovládacích prvků, pak bychom měli prvek rozšířit. V kapitole 5.1.2 si předvedeme vytvoření ovládacího prvku *DataQueryAutoCompleteTextView*, který rozšiřuje předpřipravený prvek *AutoCompleteTextView*.

Jestliže se chystáme vytvořit zcela nový ovládací prvek, pak bychom měli rozšířit třídu *View*. Tento způsob nám dává v podstatě neomezené možnosti při návrhu a tvorbě ovládacího prvku. Vytváření takového ovládacího prvku je obvykle časově náročnější, protože musíme řešit vykreslování prvku pomocí *Canvas*. V následující kapitole si předvedeme vytvoření ovládacích prvků *MonthView* a *WeekView*.

5.1.1 Kalendář

Ve vedení společnosti Kvados vznikl požadavek, aby aplikace myAvis NG umožňovala práci s kalendářem. Potřebujeme komponentu, která bude umožnovat zobrazení a úpravy událostí uložených v databázi mobilní klientské aplikace, jako jsou návštěvy, úkoly, jízdy, pohledávky apod.

Při průzkumu jsem se zajímal, zda mohu využít některý z předpřipravených ovládacích prvků operačního systému Android. Mezi předpřipravenými ovládacími prvky se nachází *CalendarView*, který slouží především pro zobrazení a výběr data. Uživatel může posouvat kalendář a dotykem vybrat datum. Pokusil jsem se zjistit, zda můžu ovládací prvek rozšířit a změnit například jeho vykreslování. Po krátkém testování jsem došel k závěru, že ovládací prvek *CalendarView* nemá smysl rozšiřovat a nevyhovuje našim požadavkům.

Systém Android mi nenabídl žádný použitelný ovládací prvek, tak jsem se pokusil najít kalendářovou komponentu vyvinutou třetí stranou. Provedl jsem průzkum a rešerši komerčních komponent. Každá použitelná kalendářová komponenta vyvinutá třetí stranou byla buď placená nebo zcela nepoužitelná. Použití placených komponent jsem se chtěl vyhnout, a proto jsem komponenty vyvinuté třetí stranou zavrhnul.

Jako další možné řešení se nabízí používání kalendářové aplikace například od společnosti Google. Kalendářovou aplikaci je možné z CRM systému otevřít jako novou aktivitu. Už tato skutečnost napovídá, že uživatelé našeho systému budou muset mít nainstalovanou nějakou další aplikaci, což může být velký problém. Kalendářové aplikace pracují s kalendářovými účty a jejich událostmi. Veškeré informace jsou uloženy v databázi systému, do které jako vývojáři máme přístup. Po krátkém zamyšlení narážíme na problém se zpětnou aktualizací událostí do CRM systému.

Řekněme, že vytvoříme úkol v CRM systému, ke kterému vygenerujeme příslušnou událost pro vybraný kalendářový účet. Pokud otevřeme kalendářovou aplikaci a některou z vygenerovaných událostí změníme, dochází ke vzniku nekonzistentních dat. Tento problém samozřejmě

má řešení. Mohli bychom vygenerované události průběžně kontrolovat a aktualizovat události v CRM systému, což by bylo velice náročné a komplikované. Další problém by nastal, pokud by uživatelé používali různé kalendářové aplikace. Možnost využití kalendářových aplikací je příliš komplikovaná a opět nevyhovující pro naše účely.

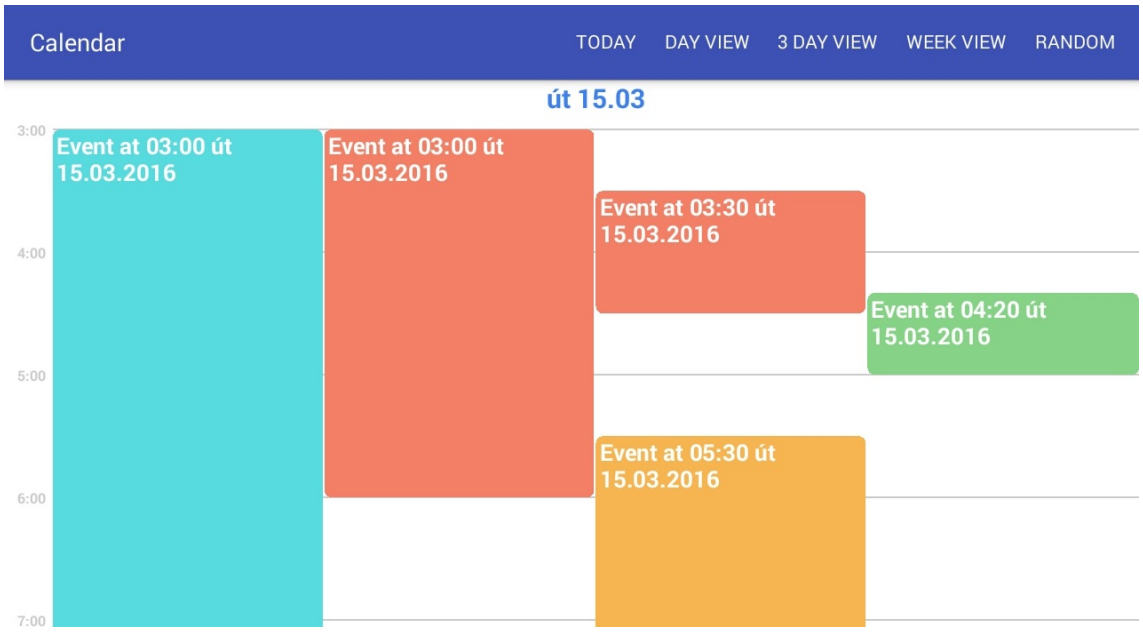
Ukázalo se, že není možné využít ovládací prvek *CalendarView*, komponenty třetích stran ani kalendářové aplikace. Rozhodl jsem se navrhnout a vytvořit zcela nové ovládací prvky umožňující práci s kalendářem. V rámci této diplomové práce jsem vytvořil ovládací prvky *MonthView* a *WeekView*, které si v této kapitole rozebereme.

Poznámka 3 Aktuální verze ovládacích prvků je graficky nedokončená, protože se čeká na konkrétní grafický návrh. Při nedávném prezentování ovládacích prvků vznikl další požadavek na možnost přetahování událostí. Provedl jsem pouze průzkum, zda lze objekty kreslené na plátno přetahovat. Průzkum ukázal, že přetahování objektů nakreslených na plátno je možné. Do konce letošního roku bych mohl získat čas na zapracování této funkčnosti do ovládacích prvků *MonthView* a *WeekView*.

Obrázek 6: Ovládací prvek *MonthView*.

MonthView, zobrazený na obrázku 6, je nekonečně vertikálně posuvný ovládací prvek, který má čtyři řádky a na každém řádku má sedm buněk. Každá z celkově dvaceti osmi buněk reprezentuje kalendářní den. Řádek buněk reprezentuje týden, přičemž pondělí je definováno jako první den v týdnu. Abychom měsíce vizuálně odlišili je každý sudý měsíc podbarvený jinou barvou. Jedna buňka umožňuje zobrazit maximálně dvě události. Pokud je událostí na jeden den

více, zobrazí se v buňce pouze jedna událost a místo druhé události je zobrazen počet dalších událostí. Při kliknutí na počet dalších událostí se zobrazí *WeekView* nastavený na vybraný den.



Obrázek 7: Ovládací prvek *WeekView*.

WeekView, zobrazený na obrázku 7, je nekonečně horizontálně posuvný ovládací prvek, který má na ose x čas a na ose y jednotlivé dny. Na obrazovce je možné vidět současně jeden, tři nebo sedm dní. *WeekView* umožňuje zoomování a tím pádem je možný i omezený vertikální posun. Při implementaci prvku jsem musel vytvořit algoritmus, který řeší vykreslování časově konfliktních událostí.

Ovládací prvky *MonthView* a *WeekView* vychází ze společné abstraktní třídy *CalendarViewBase*, která rozšiřuje třídu *View*. Zjednodušeně řečeno se jedná o ovládací prvky, do kterých můžeme plnit události a reagovat na různé dotyky prvku. V *CalendarViewBase* řešíme postupné načítání událostí, které udržujeme ve třech seznamech - předešlá, aktuální a následující perioda, přičemž perioda trvá právě jeden měsíc. Objekt *CaledarViewBase* si udržuje informaci o indexu aktuální periody a při posunování do minulosti, případně budoucnosti si seznamy předávají události, zatímco jedna perioda se vždy načte. Při načítání událostí se volá metoda *onLoad(int)* na rozhraní *IMonthLoader*, které může implementovat například aktivita nebo speciální *ViewControl*.

```
public interface IMonthLoader {
 double toPeriodIndex(Calendar c);
 List<? extends CalendarViewEvent> onLoad(int periodIndex);
}
```

Výpis 16: Rozhraní pro načítání událostí.

Událost v kalendáři reprezentuje třída *CalendarViewEvent*. Jelikož událost může trvat několik dní a zároveň potřebujeme vědět souřadnice jejího vykreslení, bylo nutné vytvořit třídu *EventRect*. Objekt *EventRect* udržuje původní celou událost, jednodenní část události a obdélník se souřadnicemi vykreslení. Následující kód 17 je metodou třídy *CalendarViewEvent*, ve které dochází k rozdělení dané události na seznam *EventRect* objektů.

```
public List<EventRect> getEventRects() {
 final List<EventRect> eventRects = new ArrayList<>();
 if (CalHelper.sameDay(m_start, m_end)) {
  eventRects.add(new EventRect(this, this));
 } else {
  CalendarViewEvent startEvent = new CalendarViewEvent(m_id, m_calendarId,
      m_name, m_location, m_start, getStartEventEndTime(), m_color);
  eventRects.add(new EventRect(startEvent, this));
  Calendar calendar = (Calendar) m_start.clone();
  calendar.add(Calendar.DATE, 1);

  while (!CalHelper.sameDay(calendar, m_end)) {
   Calendar betweenDayStartTime = (Calendar) calendar.clone();
   betweenDayStartTime.set(Calendar.HOUR_OF_DAY, 0);
   betweenDayStartTime.set(Calendar.MINUTE, 0);
   Calendar betweenDayEndTime = (Calendar) betweenDayStartTime.clone();
   betweenDayEndTime.set(Calendar.HOUR_OF_DAY, 23);
   betweenDayEndTime.set(Calendar.MINUTE, 59);
   CalendarViewEvent betweenEvent = new CalendarViewEvent(m_id, m_calendarId,
       m_name, m_location, betweenDayStartTime, betweenDayEndTime, m_color);
   eventRects.add(new EventRect(betweenEvent, this));
   calendar.add(Calendar.DATE, 1);
  }
  CalendarViewEvent endEvent = new CalendarViewEvent(m_id, m_calendarId, m_name
      , m_location, getEndEventStartTime(), m_end, m_color);
  eventRects.add(new EventRect(endEvent, this));
 }
 return eventRects;
}
```

Výpis 17: Metoda pro rozdělení události na seznam *EventRect* objektů.

CaledarViewBase si pro každou z celkově tří period udržuje jeden seznam *EventRect* objektů. Seznam se aktualizuje pouze v případě, že dojde ke změně indexu aktuální periody. Při vykreslování událostí pro kalendářní den si komponenta vybere ze seznamu *EventRect* objektů pouze takové události, které se týkají daného dne. Vybraným *EventRect* objektům komponenta nastaví obdélník se souřadnicemi vykreslení a provede vykreslení obdélníku s jeho titulkem.

Při dotyku komponenty můžeme jednoduše určit, jestli došlo k dotyku události. Stačí projít seznam *EventRect* objektů a zjistit, jestli souřadnice dotyku spadají do některého obdélníku. Jednoduchý dotyk události provoláváme na rozhraní *EventClickListener* a dlouhý dotyk na rozhraní *EventLongClickListener*. Tato rozhraní může implementovat například aktivita nebo speciální *ViewControl*. Pokud dotyk nespadá do žádného obdélníku události, je nutné na základě vykreslování daného ovládacího prvku identifikovat čas dotyku. Jednoduchý dotyk mimo událost provoláváme na rozhraní *EmptyViewClickListener* a dlouhý dotyk na rozhraní *EmptyViewLongClickListener*. V případě prvku *WeekView* určujeme datum i čas a v případě prvku *MonthView* určíme pouze datum. Třída *CalendarViewBase* má proto abstraktní metodu, kterou musí každý její následovník implementovat. V závislosti na logice vykreslování komponenty a souřadnic dotyku musíme být schopni určit datum i čas.

```
protected abstract Calendar getTimeFromPoint(float x, float y);
```

Výpis 18: Abstraktní metoda pro určení data a času.

Jak již bylo zmíněno, komponenty umožňují horizontální a vertikální posouvání. Posouvání je možné díky tomu, že přepisujeme metodu *computeScroll()* třídy *View*, udržujeme si aktuální posuv ve finální instanci třídy *PointF* a vytváříme instanci třídy *GestureDetectorCompat* a *OverScroller*.

Následující příklad 19 ukazuje přepsání metody *computeScroll()* třídy *View* a vytvoření instance třídy *PointF*. V následujícím příkladu se vyskytuje volání metody *go2ClosestOrigin()*, což je abstraktní metoda, kterou implementuje *MonthView* a *WeekView* různými způsoby. Pokud se pokusíme v *MonthView* vertikálně posunout tak, že bude viditelná pouze polovina řádku, pak volání metody *go2ClosestOrigin()* způsobí automatické posunutí tak, aby byl viditelný celý řádek. Obdobně pokud se pokusíme ve *WeekView* horizontálně posunout tak, že bude viditelná pouze polovina dne, pak volání metody *go2ClosestOrigin()* způsobí automatické posunutí tak, aby byl viditelný celý den.

```
protected final PointF m_actualOrigin = new PointF(0f, 0f);

@Override
public void computeScroll() {
 super.computeScroll();
 if (m_scroller.isFinished()) {
  if (m_currentFlingDirection != Direction.NONE) {
```

```
   go2ClosestOrigin();
  }
 }
 else {
  if (m_currentFlingDirection != Direction.NONE && scrollingShouldStop()) {
   go2ClosestOrigin();
  }
  else if (m_scroller.computeScrollOffset()) {
   m_actualOrigin.y = m_scroller.getCurrY();
   m_actualOrigin.x = m_scroller.getCurrX();
   ViewCompat.postInvalidateOnAnimation(this);
  }
 }
}

private boolean scrollingShouldStop() {
 return Build.VERSION.SDK_INT >= Build.VERSION_CODES.ICE_CREAM_SANDWICH &&
     m_scroller.getCurrVelocity() <= m_minimumFlingVelocity;
}
```

Výpis 19: Přepsání metody *computeScroll()* třídy *View*.

Při horizontálním posouvání ovládacího prvku *WeekView*, případně při vertikálním posouvání ovládacího prvku *MonthView*, vyvolává komponenta událost, která informuje o změně prvního viditelného dne. Pokud chceme na tuto událost reagovat, stačí na aktivitě nebo speciálním *ViewControlu* implementovat rozhraní *ScrollListener* a metodu *onFirstVisibleDayChanged(Calendar, Calendar)*, která v parametrech nese informaci o změně prvního viditelného dne.

Jak již bylo zmíněno, v ovládacím prvku *WeekView* bylo potřeba vyřešit vykreslení překrývajících se událostí. Pokud se překrývají časové intervaly dvou událostí, nazveme takové události vzájemně kolizní. Níže uvedená metoda 20 rozdělí seznam událostí do seznamu kolizních skupin. Kolizní skupina je množina událostí, kde pro každou událost platí, že má v dané množině alespoň jednu vzájemně kolizní událost. Skupiny potřebujeme pro určení sloupců potřebných k vykreslení překrývajících se událostí.

```
private void calcEventsPositions(List<EventRect> list) {
 List<List<EventRect>> collisionGroupList = new ArrayList<>();
 for (EventRect eventRect : list) {
  boolean isPlaced = false;

  out:
```

```
  for (List<EventRect> collisionGroup : collisionGroupList) {
   for (EventRect groupEvent : collisionGroup) {
    if (groupEvent.collide(eventRect)) {
     collisionGroup.add(eventRect);
     isPlaced = true;
     break out;
    }
   }
  }

  if (!isPlaced) {
   List<EventRect> newGroup = new ArrayList<>();
   newGroup.add(eventRect);
   collisionGroupList.add(newGroup);
  }
 }

 for (List<EventRect> collisionGroup : collisionGroupList) {
  expandEvents(collisionGroup);
 }
}
```

Výpis 20: Rozdělení seznamu událostí do seznamu kolizních skupin.

Vytvořené ovládací prvky slouží především k přehlednému zobrazení událostí v různých režimech. Události můžeme plnit do uživatelské komponenty například pomocí extenze nebo služby. Vytváření, upravování a mazání událostí je řešeno v AndroidClient aplikaci pomocí sady různých dialogů.

5.1.2 DataQueryAutoCompleteTextView

Donedávna se v AndroidClient aplikaci používala rozšířená verze *Spinneru* pro nastavení vlastností, které mají číselníkový charakter. Číselníky jsou tabulky, které mají malý počet záznamů. Časem se ukázalo, že potřebujeme ovládací prvek pro nastavení vlastností, které mají omezený a velký počet možností. Například při plánování návštěvy musíme vybrat zákazníka, u kterého návštěvu v budoucnosti provedeme. V klientské databázi mohou být staženy tisíce zákazníků a řešit výběr pomocí *Spinneru* je téměř nemožné. Pro takové případy jsem navrhnul ovládací prvek *DataQueryAutoCompleteTextView*, který je aktuálně využíván v téměř každém detailovém zobrazení.

Ovládací prvek *DataQueryAutoCompleteTextView* je rozšíření předpřipraveného ovládacího prvku *AutoCompleteTextView*, který jsem obohatil o načítání záznamů z databáze neboli entit. Jedná se o editovatelné textové pole, které při editaci uživatelem nabízí možná vyplnění. Seznam možných vyplnění je zobrazen ve vysunovacím menu, ze kterého může uživatel vybrat a tím způsobit nahrazení textu v editovatelném poli za vybraný návrh. Stisknutím tlačítka zpět můžeme vysunovací menu kdykoliv zavřít. Seznam možných vyplnění je získán z adaptéru a zobrazí se pouze v případě, že zadáme určitý počet znaků neboli překročíme určený práh.

Příklad 6

Abychom si mohli blíže objasnit další funkcionalitu, předvedu nejdříve příklad definice *DataQueryAutoCompleteTextView*.

```
<ControlItem Key="id/AutoComplete">
 <Parameters>
  <Parameter Key="DataProperty" Value="CustomerCode" ValueMode="Object"
     ValueType="java.lang.String"/>
  <Parameter Key="PersistenceService" ValueMode="TypeRef" Value="kvados.qas.
     androidclient.core.services.persistence.IPersistenceClientService"/>
  <Parameter Key="ObjectType" ValueMode="TypeRef" Value="kvados.crp.myavis.
     customers.entitymodel.MblCustomersEntity"/>
  <Parameter Key="DisplayMember" Value="Name" ValueMode="Object" ValueType="
     java.lang.String"/>
  <Parameter Key="ValueMember" Value="Code" ValueMode="Object" ValueType="java.
     lang.String"/>
  <Parameter Key="Async" Value="true" ValueMode="Object" ValueType="java.lang.
     Boolean"/>
 </Parameters>
</ControlItem>
```

Výpis 21: Příklad definice *DataQueryAutoCompleteTextView*.

■

Výše uvedený *ControlItem* má parametr *DataProperty*, jehož vlastnosti jsem vysvětlil v kapitole 4.3.12. Parametr *PersistenceService* určuje službu, která se použije pro přístup do databáze. Následující parametr *ObjectType* určuje, ze které tabulky budeme záznamy vybírat, zatímco hodnoty parametrů *DisplayMember* a *ValueMember* určují vlastnosti vybíraného záznamu. Parametr *DisplayMember* určuje vlastnost, jejíž hodnota bude zobrazena ve vysunovacím menu, a parametr *ValueMember* určuje vlastnost, jejíž hodnota bude vázána do vázaného objektu poskytnutého skrz *ControlData*.

Ve výše uvedeném příkladu vidíme, že nastavujeme vlastnost *CustomerCode* vázaného objektu poskytnutého skrz *ControlData*. V adaptéru se nachází objekty typu *MblCustomersEntity*, které se získají pomocí služby definované parametrem *PersistenceService*. Ve vysunovacím menu jsou zobrazena jména zákazníků, zatímco při výběru se do vlastnosti *CustomerCode* vázaného objektu nastaví kód zákazníka. Načítání záznamů se provádí asynchronně.

5.1.2.1 Inicializace Ovládací prvek *DataQueryAutoCompleteTextView* implementuje rozhraní *IViewControlItemControl*, proto má dostupnou instanci *ControlItemu* i s jeho parametry.

Inicializaci využijeme k vytvoření objektu *AdapterView.OnItemClickListener*, který při kliknutí na položku vysunovacího menu nastaví text v editovatelném textovém poli a uloží hodnotu, která bude vázána do vázaného objektu při ztrátě fokusu.

Následně vytvoříme objekt *TextWatcher*, který v metodě *onTextChanged(CharSequence, int, int, int)* spustí načtení entit z databáze, které vyhovují zadanému textu.

K načtení entit dochází buď synchronně nebo asynchronně pomocí *LoaderManageru*. Pokud se data mají načítat asynchronně, vytvoříme při inicializaci navíc objekt, který implementuje rozhraní *LoaderManager.LoaderCallbacks*. Objekt později využijeme při volání metody *restartLoader(int, Bundle, LoaderManager.LoaderCallbacks)* třídy *LoaderManager*.

5.1.2.2 Adaptér Ovládací prvek *DataQueryAutoCompleteTextView* má nepovinný parametr *ArrayAdapter* specifikující typ adaptéru, který se vytvoří při inicializaci ovládacího prvku. Pokud typ adaptéru není specifikován, vytváříme výchozí adaptér. Pokud typ adaptéru je specifikován, můžeme přidat další parametr *ItemLayout*, který určuje rozvržení uživatelského rozhraní pro položky vysunovacího menu. Jelikož *ControlItem* může obsahovat kolekci *ControlItemů*, můžeme načtené objekty specifikované parametrem *ObjectType* vázat pomocí adaptéru do rozvržení specifikovaného parametrem *ItemLayout*. Vázání funguje obdobně jako u *ListViewControl* a *GridViewControl* komponent, což bylo vysvětleno v kapitole 4.3.12.

Obrázek 8: Vysunovací menu ovládacího prvku *DataQueryAutoCompleteTextView*.

5.1.2.3 Fokus Pokud ovládací prvek získá fokus, dojde k načtení entit z databáze a rozbalení vysunovacího menu. Při ztrátě fokusu zkontroluje ovládací prvek položky adaptéru, které porovnává s řetězcem zadaným do textového pole. V případě nalezení shody dojde k vázání hodnoty do vázaného objektu poskytnutého skrz *ControlData*. Pokud nenalezneme položku, která

odpovídá řetězci zadanému do textového pole, dojde k nastavení výchozí hodnoty do vázaného objektu.

5.1.2.4 Načítání entit Při editaci textového pole dochází k načítání entit z databáze. Potřebujeme načíst entity, které nějakým způsobem odpovídají řetězci zadanému do textového pole. Můžeme využít stávající metody, která se používá při fulltextovém vyhledávání. V pomocné třídě *FilterHelper* existuje metoda pro přidání fulltextové podmínky do dotazu. Metodě předáme řetězec zadaný do textového pole, pole sloupců a instanci databázového dotazu.

```
addFullTextToQuery(String fullText, String[] columns, SimpleQuery simpleQuery)
```

Výpis 22: Metoda pro přidání fulltextové podmínky do dotazu.

Pole sloupců můžeme specifikovat parametrem *Columns*. Pokud parametr není specifikován, používá se hodnota parametru *DisplayMember* jako pole o jednom prvku. V následujícím příkladu je fulltextová podmínka pro případ, že zadáme do textového pole řetězec *cu* a pole sloupců má prvky *NAME* a *CUST_CODE*.

```
((A.NAME glob '*[cC][uU]*') or (A.CUST_CODE glob '*[cC][uU]*'))
```

Výpis 23: Fulltextová podmínka.

Vytvořený ovládací prvek umí plnit vlastní adaptér záznamy z databáze. Prvek umožňuje jednoduše nastavit vlastnost vázaného objektu, která má omezený a velký počet možností. Pomocí speciálních extenzí můžeme omezit množinu možností tak, že přidáme podmínky do databázového dotazu. Ovládací prvek je využíván v téměř každém detailovém zobrazení a dokonce i v jednom seznamovém zobrazení.

5.2 Implementace ViewControl komponent

Abstraktní třída *ViewControl* se používá při definování *View* ve *WorkItemu*. Obsahuje kolekci *ControlItemů* a instanci *ControlData*. Při inicializaci vytváří buď Android ovládací prvek nebo vázané rozvržení. Většinou vytváříme nové *ViewControly*, které používají původní Android ovládací prvky, jako jsou *ListView*, *GridView*, *RecyclerView*, *Spinner* apod. Rozšiřujících tříd *ViewControlu* je málo, protože se jedná o obecné komponenty, které mohou být použity na různých místech.

V následující kapitole předvedu vytvoření *RecyclerViewControl* komponenty, která má nahradit používané *ListViewControl* a *GridViewControl* komponenty.

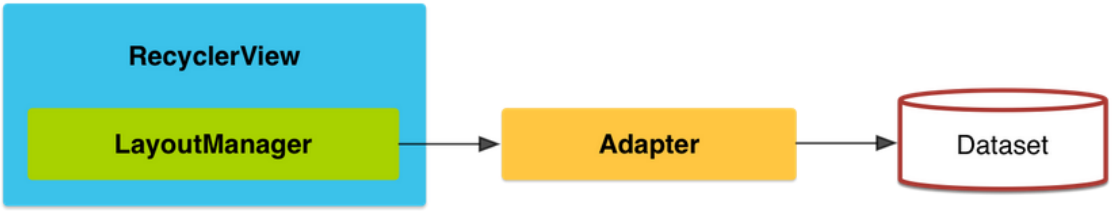
5.2.1 RecyclerViewControl

Od počátku se pro seznamové zobrazení v AndroidClient aplikaci používá *ListViewControl* komponenta. K postupnému vývoji této komponenty jsem po svém nástupu do společnosti Kvados značně přispěl. Komponenta byla dlouhou dobu bezproblémová, a proto příliš nebyl zájem o její

alternativy. Následně se ale vyskytl zásadní problém při předávání fokusu mezi editovatelnými ovládacími prvky. Tento impulz nastartoval výzkum, který mě dovedl k *RecyclerView* prvku, což je pokročilá a flexibilnější verze *ListView* prvku. Po vyzkoušení ovládacího prvku *RecyclerView* jsem začal implementovat *RecyclerViewControl* komponentu.

Ovládací prvek umožňuje zobrazovat velké datové sady. Posouvání je velice účinné, protože *RecyclerView* udržuje omezený počet *View*. Použití *RecyclerView* je vhodné, zejména pokud máme datovou kolekci, jejíž prvky se mění za běhu na základě událostí způsobených uživatelem nebo sítí.

Ovládací prvek *RecyclerView* zjednodušuje zobrazení a práci s velkými datovými sadami tím, že poskytuje *LayoutManager* a animace.



Obrázek 9: RecyclerView widget.

5.2.1.1 LayoutManager Slouží k určování polohy položek. Při posouvání určuje, kdy se opětovně použije *View* položky, kterou již uživatel nevidí. Při opětovném použití neboli recyklování *View* se postará, aby adaptér nahradil obsah daného *View* další položkou datové sady. Recyklování zlepšuje výkon, protože zamezuje vytváření nadbytečných *View*. Ovládací prvek *RecyclerView* poskytuje následující vestavěné *LayoutManagery*:

- *LinearLayoutManager* zobrazuje položky ve vertikálním nebo horizontálním posuvném seznamu,
- *GridLayoutManager* zobrazuje položky v mřížce,
- *StaggeredGridLayoutManager* zobrazuje položky v nepravidelné mřížce.

Pokud bychom chtěli vytvořit vlastní *LayoutManager*, pak stačí rozšířit třídu *RecyclerView.LayoutManager*. Při tvorbě *RecyclerViewControl* komponenty nebylo potřeba vytvářet vlastní *LayoutManager* a aktuálně používáme *LinearLayoutManager*.

5.2.1.2 Animace Projeví se například při vkládání nebo odebírání položky. Pokud bychom chtěli vytvořit vlastní animace, pak stačí rozšířit třídu *RecyclerView.ItemAnimator*. Při tvorbě *RecyclerViewControl* komponenty nebylo potřeba vytvářet vlastní animace.

Příklad 7

Abychom si mohli blíže objasnit další funkcionalitu, předvedu nejdříve příklad definice *RecyclerViewControl*.

```
<ViewControl Key="List" Type="kvados.qas.androidclient.core.ui.recycler.
    RecyclerViewControl" ControlData="Customers">
 <Parameters>
  <Parameter Key="AdapterType" Value="kvados.qas.androidclient.core.ui.recycler
      .adapter.RecyclerAdapter" ValueMode="TypeRef"/>
  <Parameter Key="ItemLayout" Value="layout/customerlistrow" ValueMode="Object"
       ValueType="java.lang.String"/>
 </Parameters>
 <ControlItems>
  <ControlItem Key="id/name">
   <Parameters>
    <Parameter Key="DataProperty" Value="Name" ValueMode="Object" ValueType="
        java.lang.String"/>
   </Parameters>
  </ControlItem>
  <ControlItem Key="id/cust_state">
   <Parameters>
    <Parameter Key="DataProperty" Value="CustState" ValueMode="Object"
        ValueType="java.lang.String"/>
   </Parameters>
  </ControlItem>
  <ControlItem Key="id/customer_group_name">
   <Parameters>
    <Parameter Key="DataProperty" Value="CustomerGroupName" ValueMode="Object"
        ValueType="java.lang.String"/>
   </Parameters>
  </ControlItem>
 </ControlItems>
</ViewControl>
```

Výpis 24: Příklad definice *RecyclerViewControl*.

■

Parametr *AdapterType* je typ adaptéru, jehož instanci vytvoříme při inicializaci komponenty. Následující parametr *ItemLayout* je jméno vázaného rozvržení uživatelského rozhraní, které vytvoříme pro položku datové sady. Pro vázané rozvržení musí platit, že obsahuje alespoň tolik ovládacích prvků, kolik je *ControlItemů*. Navíc pro každý *ControlItem* musíme v rozvržení uživa-

telského rozhraní najít ovládací prvek, který má shodné *android:id* s klíčem *ControlItemu*. Výše uvedené *ControlItemy* mají parametr *DataProperty*, jehož vlastnosti jsem vysvětlil v kapitole 4.3.12.

RecyclerViewControl vytvoří ovládací prvek *RecyclerView* a adaptér, který se stará o vytvoření vázaných rozvržení uživatelského rozhraní pro pole objektů nacházejících se v *ControlData*. Vytvořený *RecyclerViewBindingSource* se stará o naplnění adaptéru, zatímco adaptér se stará o vázání vlastností objektů do ovládacích prvků ve vytvořených vázaných rozvrženích.

5.2.1.3 Parametry Kromě povinných parametrů *AdapterType* a *ItemLayout* má komponenta další nepovinné parametry:

- *HeaderLayout* specifikuje rozvržení, které se zobrazí nad první položkou datové sady,
- *FooterLayout* specifikuje rozvržení, které se zobrazí pod poslední položkou datové sady,
- *FirstLastOffset* specifikuje odsazení, které bude nad první položkou a pod poslední položkou datové sady,
- *EndlessVisibleThreshold* určuje prahovou pozici pro načítání dalších záznamů do *ControlData*,
- *ScrollBarPosition* určuje polohu posuvníku,
- *OnInterceptTouchEventDecorator* vytvoří instanci specifikované třídy, která se použije při volání metody *onInterceptTouchEvent(MotionEvent)* třídy *RecyclerView*.

5.2.1.4 Dekorátor Dekorátory mohou změnit chování vybraných metod *RecyclerView*. Rozšířil jsem třídu *RecyclerView* a do vybraných metod jsem dodělal použití dekorátoru. Následující příklad 25 ukazuje část rozšiřující třídy *DecorateRecyclerView*, kde se v metodě *onInterceptTouchEvent(MotionEvent)* použije dekorátor.

```
public class DecorateRecyclerView extends RecyclerView
{
 //region Fields
 private OnInterceptTouchEventDecorator m_onInterceptTouchEventDecorator;
 //endregion

 //region Overrides
 @Override
 public boolean onInterceptTouchEvent(MotionEvent e)
 {
  if (m_onInterceptTouchEventDecorator != null) {
   Boolean result = m_onInterceptTouchEventDecorator.onInterceptTouchEvent(e);
```

```
   if (result != null) {
    return result;
   }
  }
  return super.onInterceptTouchEvent(e);
 }
 //endregion

 //region Nested Classes
 public interface OnInterceptTouchEventDecorator
 { Boolean onInterceptTouchEvent(MotionEvent e); }
 //endregion
}
```

Výpis 25: Část rozšiřující třídy *DecorateRecyclerView*.

Jeden z dekorátorů umožňuje vysouvat položku do stran. Při vysunutí položky se objeví definované rozvržení uživatelského rozhraní, které může obsahovat tlačítka a ikony, na které je možné zaregistrovat posluchače dotyku. Následný dotyk může vyvolat editaci nebo smazání vysunuté položky.

5.2.1.5 ItemDecoration Umožňuje dokreslit a přidat odsazení pro *View* položky datové sady. Toto může být užitečné pro kreslení oddělovačů mezi položkami, zvýraznění vybraných položek, zvýraznění seskupovacích hranic položek apod. Přidat *ItemDecoration* do *RecyclerView* můžeme pomocí metody *addItemDecoration()*. Můžeme dokreslovat před a po vykreslení *View* položky datové sady. Před vykreslením položky se volá metoda *onDraw(Canvas, RecyclerView)* a po vykreslení se volá metoda *onDrawOver(Canvas, RecyclerView, RecyclerView.State)*. Pokud přidáme do *RecyclerView* více dekorátorů, pak se volají v pořadí, v jakém byly přidány.

5.2.1.6 Adaptér Adaptér umožňuje přístup k položkám datové sady, vytváří *View* pro položky a nahrazuje obsah daného *View* další položkou datové sady v případě, že původní položka již není vidět.

Pro AndroidClient aplikaci jsem vytvořil *RecyclerAdapter* a *DragRecyclerAdapter*. Adaptér *DragRecyclerAdapter* rozšiřuje *RecyclerAdapter* o možnost tahání a odpálení položek. Tahání je překlad anglického slova „drag“ a odpálení je překlad anglického slova „swipe“. Při tvorbě adaptéru musíme rozšířit třídu *RecyclerView.Adapter* a implementovat následující metody.

```
VH onCreateViewHolder(ViewGroup parent, int viewType)
void onBindViewHolder(VH holder, int position)
int getItemCount()
```

Výpis 26: Abstraktní metody třídy *RecyclerView.Adapter*.

Následující příklad 27 ukazuje implementaci výše uvedených metod ve vytvořeném adaptéru *RecyclerAdapter*.

```
final List<Object> m_objects;

@Override
public ViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {
 int itemLayoutId = getItemLayoutId(viewType);
 View view = m_inflater.inflate(itemLayoutId, parent, false);
 return new ViewHolder(view, m_viewControl, m_context);
}

@Override
public void onBindViewHolder(ViewHolder holder, int position) {
 Map<String, Object> map = holder.getMap();
 Object newData = m_objects.get(position);
 Object oldData = map.get(AdapterHelper.DataObjectKey);

 if (oldData != newData) {
  final ViewControlItemDefinitionCollection itemDefinitions = AdapterHelper.
      getViewControlItemDefinitions(m_viewControl);
  for (ViewControlItemDefinition itemDefinition : itemDefinitions) {
   AdapterHelper.addDataToView(map, oldData, newData, itemDefinition,
       getViewControl());

   Object o = map.get(itemDefinition.getKey() + AdapterHelper.
       ViewControlItemKey);
   if (o instanceof AdapterViewControlItem) {
    ((AdapterViewControlItem) o).setDataSource(newData);
   }
  }
  map.put(AdapterHelper.DataObjectKey, newData);
 }
}

@Override
public int getItemCount()
```

```
{return m_objects.size();}
```

Výpis 27: Implementace abstraktních metod třídy *RecyclerView.Adapter*.

Adaptér udržuje položky datové sady v seznamu. Pro vytvoření instance *ViewHolder* musíme předat konstruktoru vytvořené *View*, instanci *ViewControlu* a kontext. Instance *ViewHolder* si udržuje klíčovanou kolekci, ve které je klíčem řetězec a hodnotou libovolný objekt.

```
Map<String, Object> m_map;
```

Výpis 28: Mapa v *ViewHolder*.

V konstruktoru třídy *ViewHolder* procházíme *ControlItemy*, přičemž plníme mapu klíčem *ControlItemu* a instancí ovládacího prvku daného *View*, který má shodné *android:id* s klíčem *ControlItemu*. Tato mapa je důležitá pro vázání ovládacích prvků, do kterých nastavujeme vlastnosti pomoci příslušných *ControlItemů*. Implementace poslední metody, která vrací počet položek v adaptéru je zřejmá.

Pomocí parametru *ItemTypeResolver* můžeme adaptéru umožnit vytvářet různá rozvržení uživatelského rozhraní pro položky datové sady. Stačí implementovat rozhraní *IItemTypeResolver* a přidat adaptéru daný parametr. Při implementaci rozhraní určujeme rozvržení na základě vlastností položky datové sady.

Pokud vyžadujeme, aby adaptér umožnoval tahání a odpálení položek, pak musíme použít *DragRecyclerAdapter*. Obvykle se používá dlouhý klik pro aktivování táhnutí položky, ale v AndroidClient aplikaci často využíváme kontextové menu. Vzniklý konflikt jsem vyřešil tak, že jsem implementoval funkcionalitu, která aktivuje táhnutí položky pouze při kliknutí na definovaný prvek položky. Proto má *DragRecyclerAdapter* parametr *Handle*, ve kterém specifikujeme *android:id* prvku položky, který se v rozvržení uživatelského rozhraní položky datové sady vyhledá a následně při dotyku aktivuje táhnutí položky. Zde jsou vypsány další nepovinné parametry *DragRecyclerAdapteru*:

- *DropCommand* specifikuje *Command*, který se vyvolá při dokončení táhnutí nebo odpálení položky,
- *DragFlags* specifikuje směry, kterými je možné táhnout položku,
- *SwipeFlags* specifikuje směry, kterými je možné odpálit položku,
- *SwipeLeftLayout* specifikuje rozvržení, které se zobrazí jako pozadí při odpálení položky vlevo,
- *SwipeRightLayout* specifikuje rozvržení, které se zobrazí jako pozadí při odpálení položky vpravo.

Komponenta *RecyclerViewControl* podporuje postupné načítání položek. *ControlData* mohou mít parametr, který specifikuje, po jak velkých blocích se mají načítat záznamy z databáze.

Jak již bylo zmíněno, *RecyclerViewControl* může mít parametr *EndlessVisibleThreshold*, který specifikuje práh pro načtení dalších položek do *ControlData*. Pokud se prahová položka stane uživatelem viditelná, dochází k načtení dalších záznamů z databáze do *ControlData*. Následující třída 29 ukazuje logiku volání postupného načítání záznamů.

```
public class EndlessScrollListenerComponent implements
    IEndlessScrollListenerComponent {
 //region Fields
 private ControlData m_controlData;
 private int m_visibleThreshold;
 private int m_previousTotal;
 private boolean m_isLoading;
 //endregion

 //region Constructors
 public EndlessScrollListenerComponent(ControlData controlData, int
     visibleThreshold) {...}
 //endregion

 //region Implementation
 @Override
 public void reset() {
  m_previousTotal = 0;
  m_isLoading = true;
 }

 @Override
 public void onScrolled(int firstVisibleItemPosition, int visibleItemCount, int
      totalItemCount) {
  if (m_isLoading) {
   if (totalItemCount > m_previousTotal) {
    m_isLoading = false;
    m_previousTotal = totalItemCount;
   }
  }

  if (!m_isLoading && (totalItemCount - visibleItemCount) <= (
      firstVisibleItemPosition + m_visibleThreshold) && (totalItemCount -
      visibleItemCount) > 0) {
   ((PersistedListControlData) m_controlData).executeEndlessLoading();
```

```
   m_isLoading = true;
  }
 }
 //endregion
}
```

Výpis 29: Postupné načítání záznamů do *ControlData*.

Výše uvedená třída *EndlessScrollListenerComponent* je obecná, nezná žádný ovládací prvek, proto může být použitá v *ListViewControl*, *GridViewControl* i *RecyclerViewControl*. Metoda *reset()* se volá z *RecyclerViewControl* v případě, že provedeme nový výběr do *ControlData*. Metoda *onScrolled(RecyclerView, int, int)* se volá při posouvání *RecyclerView*, jelikož implementujeme abstraktní třídu *RecyclerView.OnScrollListener*. *LayoutManager* má jednoduše přístupné všechny informace, které *EndlessScrollListenerComponent* potřebuje. Následující příklad 30 ukazuje konkrétní použití *EndlessScrollListenerComponent* v *RecyclerViewControl*.

```
protected class EndlessScrollListener extends RecyclerView.OnScrollListener {
  IEndlessScrollListenerComponent m_endlessScrollListenerComponent;

  public EndlessScrollListener(int visibleThreshold) {
   m_endlessScrollListenerComponent = new EndlessScrollListenerComponent(
       getControlData(), visibleThreshold);
  }

  public void reset() {
   m_endlessScrollListenerComponent.reset();
  }

  @Override
  public void onScrolled(RecyclerView recyclerView, int dx, int dy) {
   final RecyclerView.LayoutManager layoutManager = recyclerView.
       getLayoutManager();
   if (layoutManager instanceof LinearLayoutManager) {
    final LinearLayoutManager linearLayoutManager = (LinearLayoutManager)
        layoutManager;

    final int totalItemCount = linearLayoutManager.getItemCount();
    final int firstVisibleItemPosition = linearLayoutManager.
       findFirstVisibleItemPosition();
    final int lastVisibleItemPosition = linearLayoutManager.
       findLastVisibleItemPosition();
```

```
    final int visibleItemCount = lastVisibleItemPosition -
        firstVisibleItemPosition;
    m_endlessScrollListenerComponent.onScrolled(firstVisibleItemPosition,
        visibleItemCount, totalItemCount);
   }
  }
}
```

Výpis 30: Použití *EndlessScrollListenerComponent* v *RecyclerViewControl*.

RecyclerViewControl komponenta odstranila problém s fokusem a začíná nahrazovat *ListViewControl* a *GridViewControl*, protože je rychlejší a flexibilnější. *RecyclerViewControl* je obecná komponenta, která může být použita kdekoliv, kde potřebujeme zobrazovat seznam položek. Stačí vytvořit rozvržení uživatelského rozhraní pro položku datové sady a definovat *ControlItemy*.

6 Závěr

Zadání diplomové práce je velice obecné, protože při jeho tvorbě nebylo zcela jasné, jakým směrem se bude ubírat vývoj produktů ve společnosti Kvados.

V kapitole 1 jsem představil společnost Kvados, produkt myAvis NG, architekturu řešení myAvis a operační systém Android.

Diplomová práce je převážně věnována analýze současného stavu uživatelského rozhraní mobilního informačního systému myAvis NG. Představil jsem uvedený produkt a definoval jsem požadavky z hlediska ergonomie a komfortu ovládání uživatele. Ukázal jsem způsob zadávání dat, gesta, rozdělení obrazovky a některá typická zobrazení. V rámci této práce jsem vytvořil podporu pro více typů rozlišení a velikostí obrazovky.

V rámci diplomové práce jsem vytvořil ovládací prvek kalendář, který umožňuje denní, týdenní a měsíční pohled. Rešerši komerčních komponent jsem provedl, než jsem přistoupil k vlastnímu návrhu a řešení kalendářové komponenty. Průzkum a rešerši komerčních komponent jsem v diplomové práci neuvedl, protože jsem se věnoval převážně implementaci právě kalendářové komponenty.

V kapitole 3 jsem shrnul a popsal možnosti implementace mobilních aplikací. Poukázal jsem na výhody a nevýhody uvedených možností vývoje mobilních aplikací. Představil jsem platformu PhoneGap založenou na standardu HTML5, která umožňuje vývoj aplikací na základě jednotné technologie a jejich nasazení na více různých platformách.

V neposlední řadě jsem se věnoval návrhu nových komponent uživatelského rozhraní pro AndroidClient aplikaci. Vytvořil jsem komponenty *RecyclerViewControl* a *DataQueryAutoCompleteTextView*, které se používají v řadě zákaznických projektů. Podílel jsem se na tvorbě dalších komponent uživatelského rozhraní, jako jsou například *ListViewControl*, *VerticalTextView*, *MultiSpinner* apod. Vzhledem k tomu, že se jednalo pouze částečně o moji práci, tak jsem ji zde více nepopisoval.

Dále jsem se podílel na návrzích uživatelského rozhraní pro různé agendy aplikace myAvis NG. Působil jsem jako konzultant a software designer při návrhu uživatelského rozhraní a následně jako její tvůrce.

Navrhnul a vytvořil jsem zcela nové ovládací prvky umožňující práci s kalendářem. V rámci této diplomové práce jsem vytvořil ovládací prvky *MonthView* a *WeekView*. Vytvořené ovládací prvky slouží především k přehlednému zobrazení událostí v různých režimech. Při prezentování těchto ovládacích prvků ve společnosti Kvados vznikl požadavek na možnost přetahování událostí. Tento požadavek budu řešit během letošního roku.

Jako návrh cross-platform UI můžeme použít kapitolu 2, která se zabývá analýzou současného stavu uživatelského rozhraní mobilního informačního systému myAvis NG.

Literatura

[1] Google Inc. *Android developer* [online]. 2010-2016, [cit. 2016-04-04]. Dostupné z: <`http://developer.android.com`>.

[2] Kvados, a.s. *Kvados* [online]. 2015-2016, [cit. 2016-04-04]. Dostupné z: <`https://www.kvados.cz`>.

[3] ALLEN, Grant. *Android 4: průvodce programováním mobilních aplikací.* 1. vyd. Brno: Computer Press, 2013. ISBN 978-80-251-3782-6.

A CD-ROM

Příloha obsahuje:

1. zdrojové kódy AndroidClient aplikace,
2. kalendářovou aplikaci,
3. aplikaci generující hodnoty rozměrů,
4. ukázku definice *WorkItemu*.

A.1 Zdrojové kódy AndroidClient aplikace

Jedná se o zdrojové kódy komponent uživatelského rozhraní pro AndroidClient aplikaci. Zdrojové kódy nelze sestavit, protože je téměř nemožné je oddělit z AndroidClient aplikace do nějaké ukázkové aplikace.

A.2 Kalendářová aplikace

Jedná se o ukázkovou aplikaci, která předvádí ovládací prvky *MonthView* a *WeekView*. Události do kalendáře se generují buď náhodně nebo jsou načítány z některého kalendářového účtu zařízení. Aplikaci je možné sestavit pomocí příkazu *gradlew buildAll*. Výstupem sestavení je APK soubor, který můžete nainstalovat na zařízení s operačním systémem Android.

A.3 Aplikace generující hodnoty rozměrů

Zde je předvedena jednoduchá aplikace generující hodnoty rozměrů. Součástí přílohy je i výstup této aplikace.

A.4 Ukázka definice *WorkItemu*

Zde jsou předvedeny dvě kompletní definice *WorkItemu*, které využívají vytvořenou komponentu *RecyclerViewControl*.